(2)

CEESC-R-91-20

A THREAT-BASED

THEATER WAR DAMAGE

METHODOLOGY

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	JUNE 1991	FINAL (JANUARY 1991 to JUNE 1991)

4. TITLE AND SUBTITLE

A THREAT-BASED THEATER WAR DAMAGE METHODOLOGY

5. FUNDING NUMBERS

OMA

6. AUTHOR(S)

ROBERT H. HALAYKO

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

U.S. ARMY ENGINEER STUDIES CENTER
CASEY BUILDING #2594
FORT BELVOIR, VA 22060-5583

8. PERFORMING ORGANIZATION REPORT NUMBER

CEESC-R-91-20

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

JOINT CHIEFS OF STAFF
DEPUTY DIRECTOR FOR PLANS AND RESOURCES, J4
THE PENTAGON
WASHINGTON, DC 20318-4000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

N/A

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE;
DISTRIBUTION IS UNLIMITED

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Engineers are responsible for repairing or replacing war-damaged sustainment base facilities. Planning for the amounts and kinds of war damage repair is, however, confounded by the vagaries of war. Theater wargames generally ignore damage at rear-area installations, and war damage models typically confine their analysis to direct and collateral facility damage at one installation under one attack. The Engineer Studies Center (ESC) has used various approaches to estimate theater damage in its series of engineer assessments. In its most recent studies, ESC has developed general methodology and a PC-based computer program that extends the capabilities of the installation-level damage models to theater-level analysis. The approach incorporates scenario data and actual threat capability to estimate damage by facility, installation, and time. The primary purpose of ESC's threat-based methodology is to provide engineers a rough, but reproducible and rational, estimate of war damage for planning purposes. The accessibility of the program and the relative immediacy of results enable the user to quickly explore alternative scenarios or hypotheses. This report describes that methodology and the damage model upon which it relies.

14. SUBJECT TERMS

WAR DAMAGE; REPAIR; INSTALLATIONS; FACILITIES; MODEL; OBJECT-ORIENTED PROGRAMMING; TARGETING

15. NUMBER OF PAGES 124

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED

A THREAT-BASED THEATER WAR DAMAGE METHODOLOGY

Prepared by
Engineer Studies Center
U.S. Army Corps of Engineers

June 1991

ACKNOWLEDGMENTS

This report was prepared by Mr. Robert Halayko, Project Manager, U.S. Army Engineer Studies Center, under the supervision of Mr. Michael Kishiyama, Senior Project Manager. The methodology described in this report is a synthesis of data and methods developed by Mr. Salvatore Cremona, Major Dale Bleckman, and Mr. Richard Taylor of the Engineer Studies Center for the *Joint Operational Assessment, Engineer Requirements: European Southern Region* study. Mrs. Sally Bond provided the editorial support. ESC wishes to thank Colonel James Jenkins, Office of Joint Chiefs of Staff, for sponsoring the documentation of this methodology; Commander Robert Hood, U.S. Navy Facilities Engineering Command, for his interest in making this method more widely available; and Mr. Kevin Hager, Naval Civil Engineer Laboratories, for his assistance with the Naval Air Attack Simulation Program.

DISTRIBUTION

	No. of Copies
Joint Chiefs of Staff, ATTN: (J-4) Logistics Directorate, Washington, DC 20318-4000	2
Joint Chiefs of Staff, ATTN: (J-8) Force Resource, Structure and Assessment Directorate, Washington, DC 20318-5000	2
Headquarters, U.S. Air Force, The Civil Engineer, ATTN: AF/CE, The Pentagon, Washington, DC 20330-5425	2
Chief of Naval Operations, ATTN: OP44, Washington, DC 20350	2
Headquarters, U.S. Marine Corps, ATTN: MC-L, Washington, DC 20380	2
Department of the Army, Office of the Assistant Chief of Engineers, ATTN: DAEN-ZCM, Washington, DC 20310-2600	2
Deputy Under Secretary of the Army (OR), The Pentagon, Room 2E660, Washington, DC 20301	1
Commander in Chief, U.S. Central Command, ATTN: CCJ4/7-E, MacDill Air Force Base, FL 33608-6001	2
Headquarters, U.S. European Command, ATTN: ECJ4-LIE, APO NY 09128	2
Headquarters, U.S. Air Force Europe, ATTN: DEMRX, APO NY 09094-5001	1
HQ, U.S. Army Europe, ATTN: AEAEN-IM, APO NY 09403	1
Commander in Chief, U.S. Naval Forces Europe, ATTN: N75, FPO NY 09510-0151	1
Commander, Southern European Task Force, ATTN: AESE-EN, APO NY 09168-5125	2
Commander in Chief, U.S. Pacific Command, ATTN: J-44, Camp Smith, HI 96861-5025	2
ROK/U.S. Combined Forces Command, ATTN: CFEN, APO SF 96301-0028	1
ROK/U.S. Combined Forces Command, ATTN: CFOA, APO SF 96301-0028	1
Commander, U.S. Army Pacific, ATTN: APEN, Fort Shafter, HI 96858-5100	1
Headquarters, Pacific Air Forces, ATTN: OA, Hickam AFB, HI 96853	1

	No. of Copies
Commander in Chief, U.S. Southern Command, ATTN: SCEN, APO Miami, FL 34003	2
Commander, U.S. Army Forces Command, ATTN: FCEN-MD, Fort McPherson, GA 30330-6000	1
Commander in Chief, U.S. Atlantic Command, ATTN: J4, Norfolk, VA 23511-5100	2
Commander, Atlantic Division, Naval Facilities Engineering Command, Norfolk, VA 23511-6287	1
Commander, 412th Engineer Command (FWD), APO New York 09081	1
Commander, 412th Engineer Command, ATTN: AFKD-GCH-FW, Box 55, Vicksburg, MS 39180-0055	1
Commander, 416th Engineer Command, ATTN: CESP, 4454 West Cermak Road, Chicago, IL 60623	1
Commandant, U.S. Army Engineer School, ATTN: Directorate of Combat Developments, Concepts and Studies Branch, Fort Leonard Wood, MO 65473-6600	1
Naval Facilities Engineering Command, ATTN: NFAC-06, Hoffman Bldg, 200 Stovall Street, Alexandria, VA 22332-2301	1
Navy Civil Engineering Laboratory, ATTN: L53 (Kevin Hager), Pt. Hueneme, CA 93043-5003	1
Joint Data Systems Support Agency, ATTN: JNSLL, Washington, DC 20310-5000	1
U.S. Air Force Engineer Service Center, ATTN: DEO, Tyndall AFB, FL 32403	1
U.S. Army Concepts Analysis Agency, 8120 Woodmont Avenue, Bethesda, MD 20814-2797	1
Headquarters, U.S. Air Force Center for Studies & Analysis, ATTN: AFCSA, The Pentagon, Washington, DC 20330-5425	1
The Rand Corporation, 1700 Main Street, P.O. Box 2138, Santa Monica, CA 90406-2138	1
Defense Intelligence Agency, Bolling AFB, Washington, DC 20332	1
Commander, U.S. Army Intelligence and Analysis Center, ATTN: AIAIT-M, Building 203 (STOP 314), Washington Navy Yard, Washington, DC 20374	1

	No. of Copies
Defense Logistics Studies Information Exchange, U.S. Army Logistics Management Center, Fort Lee, VA 23801	2
Defense Technical Information Center, ATTN: DTIC-FDAC, Cameron Station, Alexandria, VA 22304-6145	2
The Pentagon Library, ATTN: Army Studies, The Pentagon, Room 1A518, Washington, DC 20310	2
U.S. Army Engineer Studies Center, Casey Building 2594, Fort Belvoir, VA 22060-5583	15
TOTAL	68

Blank Page

CONTENTS

Blank Page

ABBREVIATIONS AND ACRONYMS

AAP	Attack Assessment Program
AFCS	Army Facility Component System
AIDA	Air Base Damage Assessment Model
AR	Army Regulation
BBL	barrel
CAA	U.S. Army Concepts Analysis Agency
CEP	circular error probability
CESPG	Civil Engineering Support Plan Generator
COB	collocated operating base
COMMZ	communications zone
COSAGE	Combat Sample Generator
DAMOC	Damage Allocation Model
DOD	Department of Defense
EA	each
ESBAS	Engineer Studies Center Bomber Assessment Study
ESC	Engineer Studies Center
FEBA	forward edge of the battle area
GAL	gallon
GEOLOC	geographical location
HQDA	Headquarters, Department of Defense
JCS	Joint Chiefs of Staff
JEPES	Joint Engineer Planning and Execution System
JOPS	Joint Operation Planning System
KW	kilowatts

MOB	main operating base
MOS	minimum operational strip
NAASP	Naval Air Attack Simulation Program model
NATC	North Atlantic Treaty Organization
NCAF	nuclear capable airfields
OOP	object-oriented programming
PC	personal computer
PMD	postmortem dump
POL	petroleum, oils, and lubricants
RAM	random access memory
SEAC	Simulated Engineer Assessment of the Communications Zone Model
SOF	special operations forces
SPANS	spans
SPETSNAZ	Soviet Special Purpose Forces (SPF)
SSM	surface-to-surface missiles
U.S.	United States
USAF	United States Air Force

EXECUTIVE SUMMARY

In addition to their mission of constructing and maintaining the theater sustainment base, engineers are responsible for repairing or replacing war-damaged facilities. Planning for the expected amount and kinds of repairs, however, is confounded by the vagaries of war. Theater wargames typically ignore rear area installations, much less attempt to estimate what damage might occur over the course of a campaign. Installation-level models, that estimate the effect of individually targeted munitions and the expected resulting direct and collateral facility damage, currently exist. However, such programs are limited to one installation under attack and require too much specificity to be useful at theater-level.

The U.S. Army Engineer Studies Center (ESC) has used various approaches to this problem when performing engineer assessments of the major theaters where U.S. forces might be deployed. In its most recent studies, ESC developed a methodology that objectively addresses theater war damage. Building on the capabilities of the installation-level models, ESC formulated an approach that utilizes the best available intelligence and estimates of enemy capability to project theater damage by facility, installation, and time. This report describes that methodology and provides guidance on how it could be used and implemented elsewhere.

Much of the methodology is embodied in a computer program that ESC developed--the Damage Allocation Model (DAMOC). The program was designed to run on any PC-compatible microcomputer. It is written in TURBO PASCAL 5.5, a computer language which supports object-oriented programming (OOP). This software engineering approach is receiving much attention in computer circles, especially in the areas of modeling and simulations. ESC was able to combine its past experience using OOP with PASCAL's features to construct an efficient and extensible model. The resulting design of DAMOC proved to be a great advantage during implementation, especially when making changes and improvements to the model. Because of relatively few object-based operational models in use, software designers might find DAMOC to be of interest apart from its functional representations.

Overall, the accessibility of the system, the separation of facility damage and targeting, and the relative ease of use enable the user to utilize varying amounts of available information to estimate damage and to quickly explore alternative scenarios or hypotheses.

ESC encourages prospective users to request a copy of a distribution diskette that contains DAMOC, test data, and sample files. Such inquiries should be made directly to the Office of the Director, U.S. Army Engineer Studies Center, Casey Building 2594, Fort Belvoir, Virginia 22060-5583; phone number (703) 355-2373.

Blank Page

I. INTRODUCTION

1. **PURPOSE.** This report describes a methodology developed by the U.S. Army Engineer Studies Center (ESC) to assess theater war damage to facilities. It documents the data, design, and operation of a threat-based process to generate expected facility-level war damage at installations across a theater. In addition, it documents a damage allocation model that ESC developed as a major component of the process.

2. **BACKGROUND.** One of the more difficult aspects of war planning is assessing facility damage--what was damaged, what must be repaired, and how does it affect mission accomplishment. An air base that is fully functional may not require additional facilities to support its units. But if that base is attacked, engineers will be needed to repair runways, erect petroleum, oil, and lubricant (POL) storage, or restore operational and maintenance facilities.

a. Over the years considerable effort has been expended trying to estimate these requirements. Some models have gone to great lengths in simulating an attack and recording the damage (direct and collateral) by using explicit facility and munitions characteristics. The U.S. Air Force (USAF), in conjunction with the RAND Corporation, has developed several air base attack models: AIDA[1] and TSARINA[2]. These RAND models explored issues such as optimal attack strategy and effect on sortie generation capability. The Attack Assessment Program[3] (AAP) is another installation damage model. It is used as a front-end for damage input to the Civil Engineer Support Plan Generator (CESPG),[4] the Department of Defense's (DOD) approved engineer support model. More recently, the Navy has adapted the AAP to run on any IBM-compatible personal computer. The common trait of all these models, however, is that they tend to deal with a single installation and the effect damage has on operational capability. There is no easy way for analysts to extrapolate from individual installation damage to theater requirements.

b. ESC's interest, however, was broader and twofold--to estimate war damage for the entire theater and campaign, and to derive the resulting engineer workload. Since the mid-1970s, ESC has conducted many studies[5] that examine wartime engineer support across the entire communications zone (echelons above corps) and use different approaches. The CESPG can accept

---

[1] D. E. Emerson, *AIDA: An Air Base Damage Assessment Model*, R-1872-PR (The Rand Corporation, September 1976).

[2] D. E. Emerson and Louis H. Wegner, *TSARINA--A Computer Model for Assessing Conventional and Chemical Attacks on Air Bases*, N-2244-AF (The RAND Corporation, August 1985).

[3] Attack Assessment Program (AAP), CCTC, Computer Systems Manual, CSM UM 267-81 (1 March 1981).

[4] *Joint Operation Planning System (JOPS)--Civil Engineering Support Plan Generator (CESPG) User's Manual, Computer System Manual CSM UM 122-86* (1 April 1986).

[5] - *Bomb Damage to Critical U.S. Facilities in Europe* (ESC, May 1981).
- *Engineer Assessment, Korea: Communications Zone Analysis* (ESC, August 1987).
- *Soviet Air and Unconventional Warfare Damage, Southwest Asia* (ESC, September 1987).
- *Joint Operational Assessment, Engineer Requirements, European Southern Region* (ESC, February 1991).

direct or indirect repair tasks, but does no damage or threat calculations. When and where the damage occurs must be done off line. The Simulated Engineer Assessment of the Communications Zone Model (SEAC)[6] incorporated threat-based damage into an engineer workload model. While this general model incorporated both the calculation of damage at the facility-ordnance level and the engineer capability to repair it, it still required off-line target specification. ESC still needed a better means to generate war damage--one that would avoid the need to explicitly represent every facility in theater, and at the same time, would tie damage to threat capability.

---

[6] *Simulated Engineer Assessment of the Communications Zone Model (SEAC), Documentation and Users Manual* (ESC, June 1988).

II. METHODOLOGY

3. **APPROACH.** ESC's objective was to develop a reasonable and reproducible, threat-based system to estimate facility damage across a theater. The system that evolved was very similar to the hierarchical structure espoused in AR 5-11[7]. That regulation established an objective that components in the Army's combat model hierarchy would be able to interface. One means to accomplish this could be to use ". . . libraries of previous results from those components . . . ."[8] While the Army still awaits seamless interconnections, the linking of models of different resolution is routinely done.

a. To conduct theater analysis, the U.S. Army Concepts Analysis Agency (CAA) first uses the Combat Sample Generator (COSAGE), a stochastic division-level model, to construct a library of battle results. These "killer/victim scoreboards" record the average attrition results of replicating simulations of different posture and force structures. The scoreboards are then used by CAA's deterministic theater models as data-points in a result n-space from which new outcomes are interpolated.

b. ESC followed a similar two-phased approach. A detailed installation-level damage model is used to generate a library of attack results (damage profiles). The library is then one input to the deterministic theater damage assessment model along with scenario specific information regarding threat capability and targets. **Figure 1** portrays this methodology.

4. **DAMAGE PROFILE DEVELOPMENT.** Initially, ESC intended to use one of the installation-level models to calculate damage at each target and to aggregate the results by time and location. From available models, ESC selected the PC-based Naval Air Attack Simulation Program (NAASP)[9]. This is a stochastic model which replicates proposed attacks many times to produce an expected level of damage. However, the user must provide a laydown of installation facilities and various attributes of the contemplated attack, some of which require additional off-line calculations. ESC's attempt to use the NAASP for each installation target in a theater proved easier in concept than in practice. The problems, however, might have been fortuitous. Modifications made to ESC's initial approach resulted in a method that more equitably treats threat capability with blast and fragmentation effects. The lynchpin was the development of damage profiles.

---

[7] *Army Model Improvement Program, AR 5-11* (HQDA, August 1981).

[8] Ibid., page 1-4.

[9] J. Ferritto and K. Hager, *User's Guide for Conventional Weapons Effects Survivability Computer Programs*, UG-0012 (Naval Civil Engineering Laboratory, February 1988).

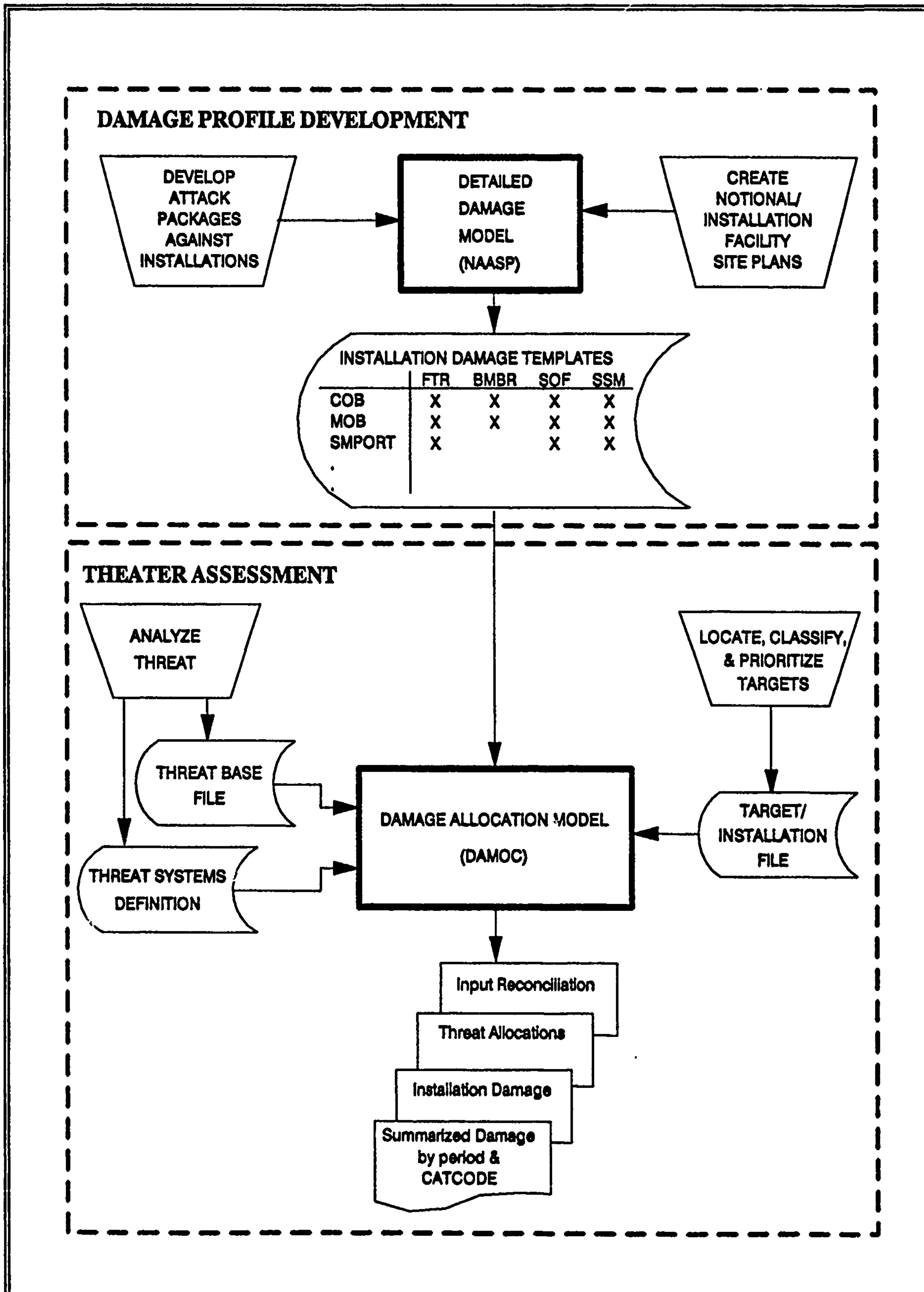

Figure 1. WAR DAMAGE METHODOLOGY CHART

a. **Notional Installations.** ESC initially planned to use the NAASP with data digitized from actual installation site plans. Each installation target within a theater would be attacked, and facility damage recorded. This ambitious approach quickly became unrealistic in execution.

(1) A variety of problems arose: installation site plans often were not available (especially for non-U.S. targets); even when site plans were available, scaling problems frustrated accurate digitization; frequently the category (e.g., construction standard, building usage) of facilities within an installation could not be established; and the time to complete one target precluded individual NAASP runs. There were also mechanical difficulties associated with converting installation data into the NAASP's digitized format. It was no easy matter to ensure that the data was formatted correctly. When an installation was digitized, numerous NAASP runs were required to refine data input until it was correct and until the program executed as intended.

(2) Based on the general unavailability of installation information and inaccuracies within the available data, ESC analysts resorted to notional installations as a way to ensure complete and consistent input data. After reviewing the classes of installation targets encountered in a theater on which information was available, patterns of typical facilities at installations of similar function were constructed; e.g., air bases have runways and communication facilities; ports have piers and storage. These were used as surrogates for actual site plans in the NAASP. The expected damage was subsequently used for all attacks against targets of the particular notional installation class. Development of the notional installations guaranteed that critical facilities were always considered and solved the building type and use problems that were encountered when interpreting real installation site plans.

(3) Development of the notional installations guaranteed that critical facilities would not be overlooked. It also solved the building type and use problems encountered when interpreting real property inventories and installation site plans. The advantages of using uniform templates for targeted installations out-weighed the probably illusory advantages of using actual site laydowns, especially when plans existed for only a small portion of the targets. Ultimately, facility laydowns were defined for 27 different notional installations.[10]

(4) The assumption that notional installations can be used throughout the theater raises an even broader possibility--if a notional installation is representative of the class of theater targets with similar functions, can this same notional design be used for similar targets in other

[10] ESC-defined notional installation classes for a Southern European scenario:

Main Operating Bases
Telecommunication Sites (fixed)
Telecommunication Sites (field)
Large Ports
Small Ports
NATO Pipeline
Ammunition Depot (fixed)
Ammunition Depot (field)
Electric Power (fixed)
Electric Power (field)
Bridge (highway)
Bridge (railroad)
Railroad Lines
Water Storage
Collocated Operating Bases
Field Camps
Hawk Sites
Large POL Installation
Small POL Installation
Tactical POL Facility
Storage Depot (fixed)
Storage Depot (field)
Highway (4-lane)
Highway (2-lane)
Tunnel (highway)
Tunnel (railroad)
Switching Yard

theaters? To a great degree, one would expect that a Southwest Asian air base would have an inventory of core facilities very similar to an air base in North America. In fact, this commonality is the underlying basis for the facility component planning systems found in each of the services. If some or all notional installations can be used across theaters, the benefit is obvious--it obviates the need to develop a completely new set of notional installations, attack packages, and damage profiles.[11] The library of profiles could then be used when a quick reaction answer forecloses any possibility of running NAASP-like analysis of actual installation attacks.

b. **Attack Packages.** Deciding on how much and what type of threat capability should be used against a target is not automatic. What level of damage is required? What threat assets are necessary to achieve that damage? The NAASP looks only at the characteristics (amount, accuracy, target point, etc.) of the ordnance and affected facilities to measure damage. The targeteer must, therefore, first determine what assets are available to cause damage.

(1) ESC defined four types of threat attack systems: fighter-bombers, bombers, special operations forces (SOF), and surface-to-surface missiles (SSM). These types then had to be defined in terms that could be used by the NAASP. The packages were engineered in reverse. First, the facilities to be attacked and the amount of damage to be achieved were specified. Then, to achieve the desired level of damage, the size and conduct of the attack was developed by trial and error means. These levels might be considered thresholds where a point of diminishing return for additional sorties has been reached (it is primarily characteristic of fighter or bomber attacks). SOF and SSM attacks were never presumed to destroy enough facilities at larger installations. (NOTE: Air, SOF, and SSM thresholds are separate. Meeting the bomber sortie saturation level will not foreclose either SSM or SOF attacks against that installation. It would, however, cutoff any additional fighter attacks.)

(2) To simplify the process, ESC standardized on an air sortie that delivered two FAB 250s. Package requirements were measured in terms of these "standard sorties." Fighter and bomber assets were similarly measured by the "standard sorties" they would provide. Thus, one SU 24 Fencer variant (external load ≤ 24,000 lbs) might be rated 3 times greater than an SU 22 Fitter (external load ≤ 7,000 lbs).

c. **Damage Profiles.** A damage profile is comprised of facilities damaged when a defined threat package attacks a specific installation. The profile defines which, and to what extent, facilities are damaged on an installation. NAASP results provide a list of facilities, amounts, and expected damage percentages, hits, and critical craters. To this information ESC adds the size of threat packages that induced the damage and the JCS category codes[12] for the various facilities. The threat size, facility information, and attack results comprise a damage profile. Profiles must be developed for each threat type, notional (or real) installation combination for which damage is to be considered in the scenario. (The user may choose not to construct profiles for unlikely uses--e.g., an SSM, with a large circular error probability (CEP), against a small highway tunnel portal.) These profiles make up the library used for theater damage assessments.

---

[11] There is no inherent obstacle to combining notional and real installation targets in a theater analysis. If good site plans exist for particular installations and time to make individualized NAASP runs is available, one should take advantage of the opportunity. The attacks against actual installations simply become damage profile templates for which only one target entry will correspond.

[12] *Department of Defense Facility Classes and Construction Categories, DOD Instruction 4165.3* (24 October 1978).

5. **THEATER ASSESSMENT.** As described in the introduction, installation-level damage models have long been available. The previously missing piece of the puzzle was the ability to go beyond installation damage models and estimate damage for the entire theater for the entire campaign. ESC's Damage Allocation Model (DAMOC) provides a solution. DAMOC is an allocation model more than a damage model because it distributes threat assets among theater targets according to defined priorities and constraints. Damage is calculated by referencing the appropriate entry in the profile library. The calculations already made in the detailed damage model are not repeated. Theater damage thus becomes a function of how threat assets can be allocated against identified targets. By focusing on available enemy capability, ESC has achieved a threat-based approach to theater damage (as opposed to the sometimes-used worst-case approach which assumes that all targets are hit). The allocation models' threat handling offers many tangible features that, up to now, have not been linked to a damage model in one trig system.

a. **Threat Sortie Manipulation.** Through the manipulation of both global and local asset-specific factors that influence sortie generation, the user can exercise a great deal of control over allocations. Specific characteristics of different threat assets must be defined, and bases or launch sites for threat assets must be identified. The ability to move assets from one base to another permits the user to tailor redeployments within, into, or out of the theater. This enables evaluation of different targeting strategies, or alignment of sorties, with estimates from more sophisticated air models. Ideally, threat sortie information should be based to some degree on the results of a high resolution air simulation. For example, in one application of its damage methodology, ESC was able to incorporate sortie and attrition results achieved by CAA.[13]

b. **Ranging.** Geographic coordinates must be entered for both targets and threat bases. The model calculates the distances between base and target to determine if target is within range of threat assets at the base.

c. **Suppression.** Attacking an air base with a full fighter package will achieve an expected level of damage. While this might render the base inoperative for a while, the possibility exists that if the "critical craters" are fixed, a minimum operational strip would be available. In recognition of this, the allocation model can be directed to attack the runway surface periodically to suppress air base operations.

d. **Data Driven.** Both the NAASP and DAMOC are data driven. The entire process, from preparing NAASP input to defining target installations, is user defined. In other words, there should be no reason why either the damage or allocation programs would have to be changed and recompiled under normal circumstances. Consequently there is no compelling need to understand how either of the models (particularly ESC's allocation model) accomplish their tasks internally. However, if a damage model other than the NAASP were used, this might not be true.

e. **User-defined Summaries.** The objective of the damage methodology is to provide expected facility damage. There are available reports on damage information at the facility (JCS category code [CATCODE]) level for each installation and on summarized damage information for specific time periods across the theater.

[13] *Engineer Studies Center Bomber Assessment Study (ESBAS)*, CAA-MR-90-47 (U.S. Army Concepts Analysis Agency, September 1990). The study relied on the COMO Integrated Air Defense Model to provide the Corps of Engineers the number of enemy bombers that can reach air bases This information was then used to generate emergency war damage repair requirements.

f. **PC-Based.** One of the advantages of ESC's approach is that it can all be done on a PC-compatible microcomputer. To further maximize program execution capability, most input in the allocation model is saved in dynamically allocated memory locations, rather than fixed arrays. The accessibility and general capability of the methodology facilitates use. It also increases the likelihood of experimentation and alternative evaluations. The ubiquitous PCs also guarantee portability.

6. **ASSUMPTIONS.** Despite our best intelligence gathering efforts, when war starts no one can predict how an enemy will choose to attack U.S. and allied bases. Munitions effects can be modeled with great accuracy if we know the aim point and the proximate facilities. But how confident can one be that the munitions get to the target, or that the target has even been selected by the enemy. The situation is analogous to the Heisenberg Uncertainty Principle--the greater our quest for accuracy, the greater our associated error. With that in mind, ESC made several assumptions in confecting its methodology.

a. **Threat Assignment.** It is conceivable that intelligence means might obtain enemy attack plans prior to hostilities and know exact targeting information. But once that attack begins, attrition, maintenance, counterattacks, mission success, forward edge of battle area (FEBA) movement, etc. make it difficult to estimate what would happen in the following days, much less predict events weeks or months later. ESC concluded that it is impossible to predict these events with any certainty. The best compromise is to adopt a consistent and reproducible method that can be manipulated easily to examine different assignment schemes.

b. **Sortie Equivalence.** The damage model used by ESC to assess installation level damage did not concern itself with how munitions got to a target--its needs were for munitions attributes (fuzing, aiming errors, etc.), not the performance specifications of the delivery platform. To reduce complexity and situations for evaluation, ESC standardized using one conventional munition.[14] This meant that only one fighter bomber configuration needed to be defined. That definition would be in terms of the number of those munitions it could carry. For example, suppose the nominal weapon pattern/load is 4 standard bombs. If a SU-24 Fencer carries 4 bombs and a MIG-27 Flogger carries 6, then each Fencer contributes 1 standard sortie, while each Flogger is worth 1.5 for sortie capability purposes.

c. **Allocation Rules.** Deciding how many attacks should be made against a target is a function of several factors: type of installation, type and amount of facilities, number of available attackers, amount of damage from prior attacks, and the priority of the target. ESC adopted a straight-forward rule that considered these factors during targeting. ESC assumed that it was better to apply reasonable criteria consistently, than to try to intuit the thoughts of threat planners. As a compromise, ESC settled on defining attack packages whose expected results would

---

[14] The Soviet FAB-250 General Purpose Bomb with instantaneous fuze. See *Red-on-Blue Manual (Effectiveness Estimates for Soviet/Warsaw Pact Nonnuclear Munitions)(U)*, 61 JTCG/ME-77-15 (Rev. 1, 1 October 1982, Change 2--30 April 1986).

achieve the desired level of damage. Allocation was made according to target priority. ESC assumed a simple, preemptive priority rule: the value of a target installation corresponded to its relative location in the theater target list.[15] For example, the third target in the list would be attacked, if possible, before the fourth.

d. **Proportionality.** ESC associates a certain number of threat assets with a desired level of damage. Once sufficient threat assets are directed against a target, the model will look to target installations of lower priority. Frequently, available sorties will fall short of required attack levels. Rather than use an "all or nothing" strategy, ESC's methodology allocates what it has available. In the damage model, attacks are directed against specific facilities. If only half of the required number of sorties are allocated, then one would expect that only half of the targeted facilities will be hit. Since ESC does not know which half of the facilities were hit, damage and hits to facilities are prorated according to the proportion of sorties actually sent and the amount needed for the desired level of damage.

---

[15] The user should be aware that there is no attempt to optimize sortie allocation with regard to coverage. Like targets, threat assets are allocated sequentially. The program does not look at all assets to find the ones whose range comes closest to the distance to the current target. Therefore, the user should list threat assets in the THREAT file according to range--shorter range assets at the beginning, longer range at the end.

Blank Page

III. APPLICATION

7. **INPUT.** The two-step damage process (explicit installation damage, theater allocation) raises the need for two sets of input. The user should refer to the NAASP documentation (or to appropriate references if another damage model is used) for its input requirements to produce the damage profiles. DAMOC data requirements are briefly described below. More definitive descriptions appear in Annex A.

a. **Threat Assets.** Threat data drives the allocation model. Nothing happens unless attacks can be made. The most important threat asset data element is its type. The attack packages derived from the NAASP use four threat types: fighters, bombers, SOF, and SSM. Threat assets are the actual systems derived from intelligence and planning sources for which a type must be designated. In addition to the type, a user can define up to three theater movements, as well as performance, attrition, and availability data for each threat asset. Distinctions are drawn among different rates for different types; therefore, the user is advised to consult the associated descriptions found in Annex A.

b. **Threat Bases.** To facilitate management of threat assets, they must be associated with individual bases. While this refers primarily to threat air bases, it can, however, be broadly viewed to include SSM launch points and tactical helipads used for SOF insertions.

c. **Notional Installation Classes.** The allocation model uses installation damage data produced by the detailed damage model. For each class of notional installations, damage profiles are given for defined threats. Note that an installation class need not have profiles for all possible threats. For example, performance limitations might indicate that SSM accuracy precludes use against bridges. The exclusion or inclusion of particular threat installation profiles can be used to control allocations.

d. **Targets.** This file lists the name, installation class, and location of all targets to be considered. Most targets are fixed installations. Since continuous targets such as roads and pipelines do not have discrete saturation levels, the user can designate those targets for continuous attack (i.e., they cannot be saturated). Likewise, the user can define mobile military targets that, if attacked, may not generate engineer repair requirements, but will divert sorties from other missions.

e. **Scenario.** While some explicit scenario information is built into the threat file (e.g., attrition), the other controlling data are entered at the beginning of program execution. The parameters that decide how the model will operate are information such as duration of simulation, frequency of reports, countries or organizations to be reported, suppression frequencies, and names of input files.

8. **OUTPUT.** DAMOC provides a full range of useful reports and messages. More definitive descriptions appear in Annex B.

a. **Data Summaries.** At the beginning of DAMOC's execution the program reads threat base, threat, damage profile, and target files. In addition to converting data elements into internal textual and numeric formats, the program checks data validity: threat groups cannot be assigned to non-existent bases, targets must have a valid installation type, damage profiles within a notional installation must be consistent across facility sizes, etc. In constructing an application, the user should review the data summaries to assure that intended entries are accepted.

(1) ***Threat.*** Threat information is entered in the base and threat asset files. Base files identify valid threat locations from which attacks originate. Although the user can enter a base's full name, the identifier used internal to DAMOC is the code identification. It is this code that is checked against entries found on the threat asset files. If a nonexistent base is encountered, the asset entry is rejected.

(2) ***Damage Profiles.*** The installation profiles, derived from the detailed damage model (or models), are read and assembled into a damage profile table. While building the reference table, DAMOC culls all the category codes encountered and lists them. It also summarizes the damage table by providing the notional installations that have been encountered; the threat types that can be used against them; the number of catcodes comprising each profile; and the size of the associated threat packages. It also reports when an internal check on the data fails from either a facility inventory inconsistency or an unknown threat type.

(3) ***Targets.*** Target installations are reviewed against the profiles found in the damage table. A target's reference type must correspond to a defined notional installation type. Country codes are not checked--the user must define the country field depending on the problem's demands.

b. **Facility Damage Summaries.** Periodically during DAMOC's execution a summary of damage for all selected installations is printed. It shows the extent of each facility in the installation subset and the associated damage, hits, and craters that occurred during the period. The user may designate a subset of installations for the summary reports (this subset will also decide which installations will appear in the installation summaries). The designation uses the country codes found in the target file. The reporting depends on how the codes were initially defined and suggests that some thought should be given to their initial definition. While the obvious use is to designate nationality, one could also use it to discriminate among U.S. facilities in different nations, services, or major commands (e.g., "U" = U.S. installations; "T" = Turkish installations; but "t" = U.S. installations in Turkey). Note that the facility totals represent only those found in installations in the report set. It should also be emphasized that the report set has nothing to do with targeting. Given the same threat and target input, sorties allocation will be identical, regardless of the report selection.

c. **Installation Summaries.** The facility summaries are convenient to get a general idea of attack intensity. By itself, however, it would be of little use to engineer planners. The individual installations provide the planner with an idea of what, when, and where engineers will be needed. The report breaks out facility damage, facility hits, and critical crater percentages by time periods. It also shows when attacks were made. The damage can be used in engineer workload models to assess the adequacy of engineer capability.

d. **Sortie Log.** A log file is created by DAMOC to record all sortie allocations, unused assets, and threat changes (i.e., redeployments) during the scenario. This file is only intended to enable the user to confirm that sorties and movements occurred as expected.

9. **PROGRAM DESIGN.** This section is a quick overview of the damage allocation (DAMOC) program. The success of DAMOC[16], with respect to extensibility and execution speed, is largely attributable to its design. It uses a software technique called object-oriented programming (OOP).[17] This enhances the ability of the modeler to decompose a problem. In more traditional program languages (e.g., FORTRAN) the programmer must represent the model using only a few data structures (integer, real, and alphanumeric variables). OOP languages enable the programmer to define additional structures, which can be problem specific. In DAMOC there are types for installations and profiles, as well as types for integers and strings. Studied decisions on how to define object types will greatly influence how well a problem can be modeled. The interested reader is referred to Annex C where the individual program elements are described more fully.

a. **Object Hierarchy.** One feature of OOP enables the programmer to build or extend previously defined objects. DAMOC's general structure has three layers. The first layer defines data structuring classes that are used extensively by other objects in the model. The middle, and by far the largest, layer contains the object classes that define the methods and elements that comprise the threat-installation-damage context. The third layer is the main program that uses the object structure to simulate theater damage results. **Figure 2** shows this stratification.

b. **Unit SIMSETx.** One of the benefits of OOP is the memory utilization derived from tighter control over data structuring. Rather than defining large arrays (which either constrain the number of entries or are purposely too large) to retain information, as FORTRAN would require, the programmer can use objects to request only as much memory as needed, as well as to encapsulate data. When an object is dynamically created, a way must be preserved to reference or "point" to it, otherwise the program has no way to make use of it. One device used extensively in DAMOC is the two-way list. Such lists are realized in the model by employing derived types of *HEAD* and *LINK* objects. First, the list must be created (*new HEAD*), and then objects can be added to the list (*LINK.into(HEAD)*). Various list functions are defined for both *HEAD* and *LINK* objects (and consequently for objects derived from them). Such methods designate the first or last object in the list; indicate whether the list has any objects, or is empty; enable an object to be put in, or taken out of, a list; and count how many items are currently in the list. When, for example, a new ***installation*** is created (defined as a derived object of *LINK*), it is placed in an ***area*** object (defined as a derived object of *HEAD*). The program can then search through ***area*** to access that ***installation***. This list device is a common structure in OOP languages.[18]

c. **Unit COMMZ: Object Classes.** The structure of DAMOC can be viewed as a collection of different objects that have certain attributes and procedures. Separately, the objects should represent a reasonable decomposition of the problem environment. Together, they should provide a substrate upon which an application can be built. The attributes describe the state of the object, and the procedures define how interactions between or among objects occur.

---

[16] Compared to some other possible approaches, DAMOC was quite efficient. Initially, ESC contemplated using a spreadsheet-based methodology. The danger of using spreadsheets on the wrong problem was dramatically illustrated. That approach was marked by inflexibility, constant human attention, mushrooming storage demands, and completion times measured in days, possibly weeks. DAMOC did it all by using a fraction of the storage needs, by reducing data to manageable levels by eliminating needless deviations, by requiring little more than a few data parameters from the user, and by executing in seconds on the very same machine.

[17] DAMOC was written in TURBO PASCAL 5.5, a dialect of PASCAL that incorporates true object-oriented capability.

[18] See explanation of CLASS SIMSET in *Introduction to Simula 67*, Gunther Lamprecht (Friedr. Vieweg & Sohn, 1983).

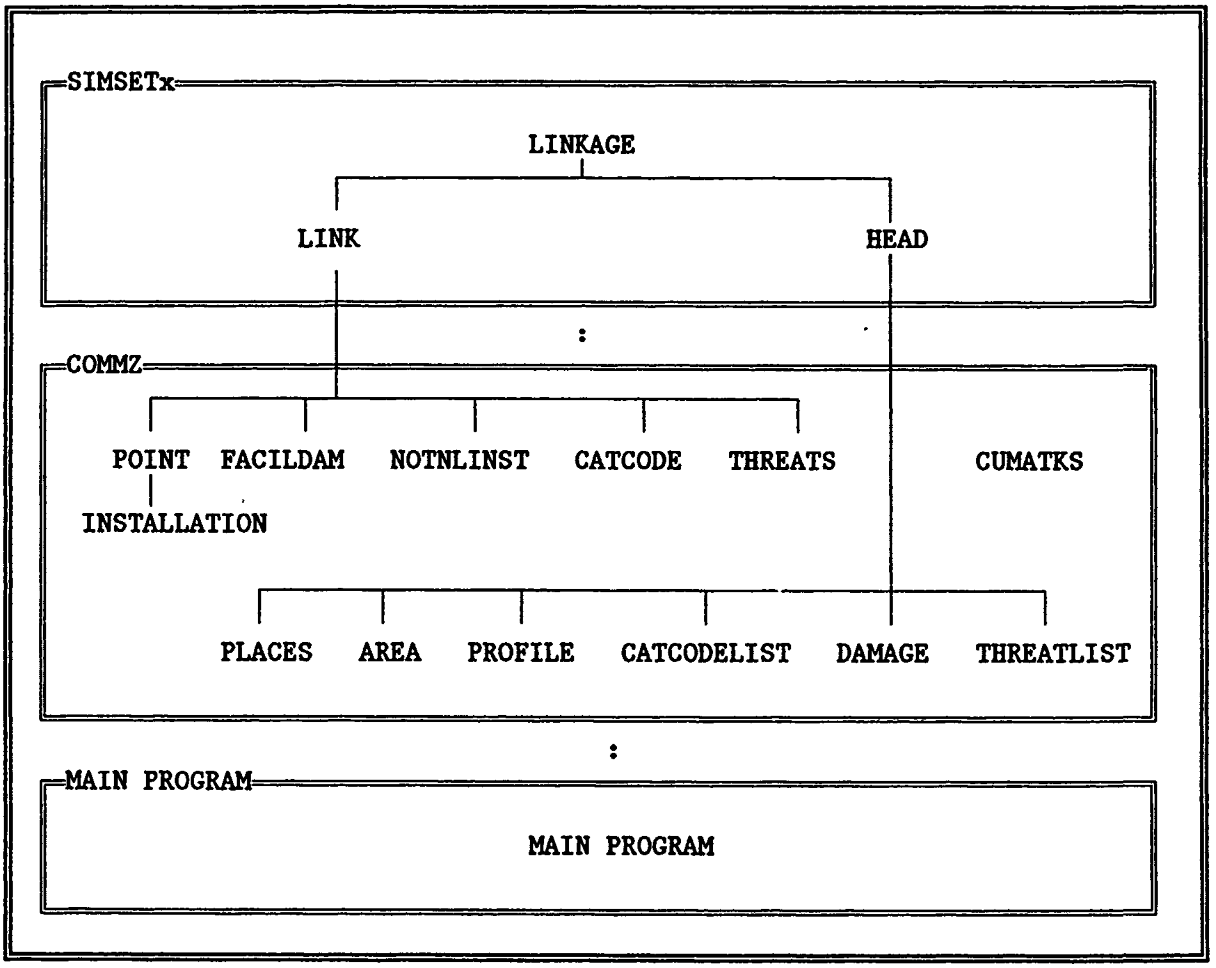

Figure 2. OBJECT CLASS HIERARCHY OF DAMOC

d. **Main Program.** After defining the elements, or objects, that comprise the theater damage environment, their behaviors must be orchestrated. The main program initiates the creation of the scenario components, decides what threat assets are available and where they should attack, and collects data for or initiates the various reports.

10. **MODELS EXECUTION.** The damage methodology is a two-stage process. Before DAMOC is run, the user must decide what damage profiles are required. If existing profiles are sufficient, there may be no need to generate new profiles--it may be enough to merely create a few special installations. (While consistency may weigh heavily on a decision, it is not necessary to use the same detailed damage model for all the profiles.) After the needed profiles have been revised or created, the user must assemble the necessary threat and target information required by the damage allocation model. Below are some general comments about execution characteristics of NAASP and DAMOC.

a. **NAASP.** Being PC-based is the greatest advantage of the Navy damage model. Having ready access to the program allows a user to explore input variations and their affect on output. The only special hardware requirement is the need for a math co-processor.

(1) *Operation.* The NAASP has a menu-driven data preparation module which provides an interactive session to build the various input files (target, weapons, damage, plot, and attack). Different portions of the model can be run separately, or all the input can be combined to run an entire case.

(2) *Environment.* The NAASP contains support logic for certain optional hardware peripherals that facilitate using the system. Unfortunately, ESC did not have one of these items--a compatible digitizer. This meant that much of the site plans had to be manually entered--a frustrating task.

b. DAMOC. Like the NAASP, DAMOC was designed to run on any PC/AT or PC/AT-compatible. No special hardware requirements are necessary to run the program. The only caution is in the area of security. Because threat and target input are likely drawn from classified sources, local security limitations would have to control the execution environment. Annex A describes DAMOC input in detail. Some of the operational characteristics of interest to potential users are listed below.

(1) *Interactive.* The program is designed to be run interactively. A series of questions are posed to which the user must respond before the program will go forward. In the interactive mode, all output goes to the screen, except for threat dispositions written to the SORTIE.LOG file.

(2) *DOS Redirection.* While it may be useful to run DAMOC interactively to see how things proceed or what errors might be uncovered in the input, the amount of information that appears on the screen will overwhelm a user. To capture this information, one can use DOS's file redirection feature. The normal query-response cycle can be bypassed by entering the following command:

```
DAMOC < control.file > output.file
```

The *control.file* contains responses to the questions posed during interactive processing. The *output.file* will receive all the data that would otherwise go to the screen. Note that it is not unusual for output files to require 300, 500, or as much as 1,000 kilobytes of storage (the number of installations is the controlling factor). The user should keep this in mind when designating destination files (a hard drive or Bernoulli-like removable disk may be necessary).

(3) *Specifications.* DAMOC currently runs comfortably on a standard AT machine with 640 kilobytes of random access memory. Execution speed is a function of scenario length, report selections, number of threat assets, and number of targets. Run times on 80286-based machines have ranged from a few minutes to several hours. A test run on an 80386-based PC-compatible saw immediate three-fold execution time improvements. The number of lines of code for the three program units in DAMOC (SIMSETx, COMMZ, and DAMOC) together total less than 1800 lines of code. The memory requirements for the associated symbolic files are less than 60 kilobytes. The executable element (DAMOC.EXE) is less than 40 kilobytes. This last value should not be misinterpreted; the executable size refers only to code. The actual memory requirements is a function of the input data. ESC has run scenarios that use close to 400 kilobytes of random access memory (RAM) for data. Even at that, the model runs comfortably within the 640-kilobyte DOS address space.

(4) *Limitations.* ESC has tried to make the model as unrestrictive as possible. Nonetheless, there are several internal parameters of which a user should be aware:

- there are only 4 threat types--fighter, bomber, SOF, and SSM
- 3 changes in threat rates or redeployments can be made
- 75 facility categories can be tracked
- scenarios can be up to 180 days long
- the number of time periods must be less than or equal to 10 (i.e., length_of_scenario/length_of_period ≤ 10)

Increasing any of these parameters, except the threat types, requires nothing more than changing several internal dimension statements. Changes to threat types have much larger implications and necessarily can be accomplished only after making substantial changes to the model.

IV. SUMMARY

11. **FUTURE ENHANCEMENTS.** As ESC applies its damage methodology, modifications and improvements continue to be made, particularly to DAMOC. One of the advantages of the allocation model is its receptiveness to change. It has proven to be highly extensible. Based on discussions and experience, ESC foresees the following modifications being made to refine the allocation model and further enhance its utility.

a. **Installation Modularization.** Presently the model deals with 27 classes of notional installations. For representational and targeting robustness, it might be desirable to visualize installations as groups of sub-installations. An air base might have runway; petroleum, oils, and lubricants (POL); maintenance; and other facility subsets. By supporting a certain amount of modularization, DAMOC could adopt more selective targeting than the currently-used installation priorities. This approach might be more imperative if smart munitions were included and used against facilities rather than installations.

b. **Threat Types.** The use of only four threat types may be restrictive, especially in reducing fighters and bombers to common units. Aircraft are currently standardized on one type of munition. While this simplifies the process and reduces the number of NAASP cases, it would be more realistic to consider several munition types (conventional and smart) and the carriers that can or cannot deliver them. Although this would require a few internal changes to DAMOC, the real impact would be on the analyst having to make that many more preparatory runs of the detailed damage model to develop the various attack package-facility damage sub-tables.

c. **Reconstitution (Implied Engineer Capability).** DAMOC currently has a global switch that resets damage.[19] It was included under the premise that U.S. and indigenous engineer capability might be able to reconstitute (i.e., repair all damage) an installation. This is different in degree from the need for suppression that contemplates selective repair (in particular runway craters). The all-or- nothing impact of "toggling" reconstitution across all installations seems too broad in retrospect. Clearly, it would be more desirable to selectively reconstitute installations based on knowledge of local engineer capability and the time required to effect repairs. It would be relatively easy to modify DAMOC so that reconstitution can occur at designated installations. However, it is difficult to determine when and where reconstitution should occur because there is no explicit engineer representation in DAMOC.

d. **Threat Ordering.** As noted in paragraph 6c, the user should be cognizant of the sequential nature of sortie allocation. It would be an easy task to have DAMOC order the threat according to range, with the option of disabling that feature if theater geometry reduces its importance. (If necessary, an "assignment problem" algorithm could conceivably be incorporated. This would probably have, however, major execution and memory implications.)

[19] See Annex A discussion of Run Control File elements.

e. **CESPG/JEPES Linkage.** ESC's war damage approach is an adjunct to theater planning. Calculating where, when, and how much damage occurs is usually preliminary to determining if planned engineer capability is adequate. Since capability must also be applied to new construction requirements, damage and construction should be addressed together. One obvious way to do this is to use DAMOC results as input to the CESPG (or its eventual successor--the Joint Engineer Planning and Execution System (JEPES)). DAMOC could be modified to produce damage information in a mutually compatible format.

12. **ASSESSMENT.** ESC's damage methodology espouses a pragmatic approach to the insoluble problem of predicting war damage. Its primary purpose is to provide engineer and military planners with a reasonable estimate of potential theater-wide war damage. The estimate couples output from facility damage simulations with scenario-dependent factors--threat capability, target priorities, and campaign factors. As such, the approach extends rather than replaces current damage models by framing the amount of damage within the context of theater threat capability. Other attributes of DAMOC--its modest size and its PC compatible implementation--make it highly portable. To encourage potential users to evaluate and hopefully utilize the methodology, ESC will provide, upon request, a distribution disk containing an executable version of the allocation program (DAMOC.EXE), program files, and sample data. This is enough to make a test run and observe the execution time and ease of use. To obtain this disk, contact the Office of the Director, U.S. Army Engineer Studies Center, Casey Building 2594, Fort Belvoir, Virginia 22060-5583; phone number (703) 355-2373. (NOTE: For a copy of a detailed damage model such as the NAASP, a user would have to contact the organization responsible for its development.) Overall, the accessibility of the system, the separation of facility damage and targeting, and the relative ease of use enable planners to adapt to varying amounts of available information to estimate damage and quickly explore alternative scenarios or hypotheses.

LAST PAGE OF MAIN PAPER

ANNEX A

DAMOC INPUT

ANNEX A

DAMOC INPUT

1. **PURPOSE.** This annex describes the input file formats and data used by the Damage Allocation Model (DAMOC).

2. **SCOPE.** The annex is limited to input discussions for DAMOC. Insofar as DAMOC is a data-driven model, this might also be viewed as a user's guide. The companion to this annex would be a description of input for whatever detailed damage model is used, if additions or alternative damage profiles are necessary. For such information, the user should consult the applicable user's manual.

3. **DESCRIPTIONS.** The scheme used to define the data uses both examples and textual descriptions. First an extract or portion of the file is shown. That is followed by a field format definition showing character and field positions. (NOTE: Character or string entries should be left justified in their subfields because leading blanks are not stripped out.) Finally a brief description of individual datum is provided along with desiderata that should be heeded while constructing the files.[1]

a. **Threat Bases File.** This file defines the locations from which various attacking forces originate and fixes the location (latitude and longitude) at which the attack starts. It is used to calculate whether particular attacking types are within range of specified targets. This file can also be used to define locations to which threat assets will withdraw or forward deploy.

(1) *Formats.* **Figure A-1** below is an example of the Threat Bases file and the file format.

```
Example:  Threat Bases File

airbase number 1          base0          100000N   0500000E
airbase number 5          base5          150000N   0600000E
Alpha                     base2          110000N   0520000E
Beta                      base3          120000N   0540000E
IV Corps                  base4          130000N   0560900E
Capital                   base6          140000N   0585000E
Upsilon                   base1          150000N   0580200E


Format:  Threat Bases File

         1         2         3         4         5         6         7         8
12345678901234567890123456789012345678901234567890123456789012345678901234567890
          |              |              |        |

     Base Name      Base Code      Latitude Longitude
```

Figure A-1. THREAT BASES FILE EXAMPLE

(2) *Explanation.* **Figure A-2** provides the definitions of Threat Bases file input. Several things about the Threat Bases file are worthy of mention. First, the model **does** consider hemisphere; therefore, latitude *V* must equal *N* or *S* and longitude *H* must equal *E* or *W*. Second, although the base code is user-defined, a standard code, such as geographic location (GEOLOC), is recommended where possible. Third, by purposely defining bases well beyond the range of threat systems, one can, by using the redeployment entries for threat systems, simulate movement into and out of the theater of operations.

---

[1] File sizes are not restricted. There is no specific limit on the number of records contained in the data files. To do this, DAMOC exploits the dynamic memory facility of PASCAL 5.5 (objects are dynamically allocated on the *heap* rather than on the *stack*). While ESC has defined some rather large scenarios (120 days; 15 bases; 20 threat systems; 500 targets), the *heap* has not come close to being used up. It is not, however, inexhaustible, and in the event that it is exceeded, the system will lock up.

Input Item	Start Col	End Col	Description
Base Name	1	20	Name of location from which threat attacks will originate
Base Code	25	29	Code (≤ 5 characters) assigned to threat base location (used for threat placement and moves)
Latitude	41	47	Latitude expressed in: ddmmssV
Longitude	51	58	Longitude expressed in: dddmmssH

Figure A-2. DEFINITIONS OF THREAT BASES FILE INPUT

b. **Threat File.** This file provides the scenario dependent description of how the threat will operate against the targets. At designated times, groups can be moved from base to base to correspond to scenario-based movements. Operational, casualty, or expenditure rates can also be designated and can be made both group and time specific.

(1) *Format.* Examples of a Threat file and the file formats are found in **Figure A-3** on the next page.

```
Example:  Threat File

  FLOGGERS               FIGHTER   base0 1281   50   20    6base1          <--type 1 record
                   1  .90  .10                                             <--type 2 record
  BADGERS                BOMBER    base3  750   20   10    3base9
                   5  1.0 .10   10  .70  .20
  SPETSNAZ DIV           SOF       base0  150   20    0
                  10  .10  .10
  SPETSNAZ CORPS         SOF       base4  300   20    0
                   5  .15  .15
  SCUD                   SSM       base6  400   50    0  15base0
                  12  1.   .20

Format:  Threat File  [record type 1]

           1         2         3         4         5         6         7         8
  12345678901234567890123456789012345678901234567890123456789012345678901234567890
  |                    |      |    |    |    |    |    |    |    |    |    |
  Threat Description   Threat Start Rng  Amt  Amt  Move Base Move Base Move Base
                       Class  Base       Beg  Min  day1      day2      day3

Format:  Threat File  [record type 2]

           1         2         3         4         5         6         7         8
  12345678901234567890123456789012345678901234567890123456789012345678901234567890
              |    |    |    |    |    |    |    |    |
             day  rate rate day  rate rate day  rate rate
              1   1/1  2/1  2    1/2  2/2  3    1/3  2/3

  day #d:= day at which rate change occurs
  rate type/#d :=  for planes rate 1 = operational ready rate
                              rate 2 = casualty/loss rates
                   for SSM    rate 1 = availability (0 or 100)
                              rate 2 = usage rate
                   for SOF    rate 1 = pre target attrition
                              rate 2 = recovery losses

NOTE:  The days associated with movement and rate changes need not be the same.
```

Figure A-3. THREAT FILE EXAMPLE

(2) *Explanation.* Two things should be considered when assembling the threat system file. First, threat assets are allocated each day according to the order in which they were initially entered. Second, beginning amounts may not be the actual number of available aircraft or units. In response to the former, the user should probably put short-range systems at the beginning of the file and long-range assets near the end of the file. In regard to beginning amounts, the important thing to remember about threat types (e.g., FIGHTER, SOF) is that they must be counted in terms of standard units. The model has no internal knowledge of threat organization or configuration. If SOF teams simulated in the detailed damage model (e.g., the NAASP) were 20-man teams, than a SPETSNAZ Brigade would be defined by the number of such teams it controlled. The units can also vary within a platform: one FLOGGER might have a normal sortie capability of 2 units, but a long-range "B" version (range increases because external tanks are used) would only be worth "1." Also, there is no implicit correlation between "move" and "change" days. These values need not correspond to each other--they are to simulate

events and conditions in the controlling scenario. It should be emphasized, however, that there are no default rates or presumption of availability on day 1. Consequently, there must be an entry in "change day 1" and associated rate. (NOTE: "change day 1" can indicate any day in the scenario; it should not be interpreted as meaning day=1.) Definitions of Threat File input for records 1 and 2 are found in **Figures A-4 and A-5** respectively.

Input Item	Start Col	End Col	Description
Threat Description	1	20	Name of associated threat group. It might indicate weapon type and organization: 5th SPETSNAZ Bde or Fencers/1st Air Wing
Threat Type	21	30	Type of threat asset: FIGHTER, SOF, BOMBER, or SSM
Base (starting)	31	35	Starting base
Range	36	40	Range of threat system (nautical miles)
Beginning Amount	41	45	Starting number of assets. (Note that this is not necessarily a count. Plane sorties must be in notional sortie terms; SOF counts should be multiples of nominal group size.)
Minimum Amount	46	50	Lowest level that asset can reach. (replacement pipeline)
Move day #1	51	55	Day on which 1st asset redeployment occurs from starting base to next location
Base	56	60	Base code of new location
Move day #2	61	65	(see above)
Base	66	70	(see above)
Move day #3	71	75	(see above)
Base	76	80	(see above)

Figure A-4. DEFINITIONS OF THREAT FILE INPUT (RECORD 1)

Input Item	Start Col	End Col	Description
Change Day 1	11	15	Day at which initial rates are active (if > 1 then asset considered initially unavailable)
Rate#1 for day 1	16	20	Value of first rate#1. (NOTE: Rate#1 is interpreted differently for each threat type: for FIGHTER and BOMBER, it is the operational readiness rate (decimal); for SOF, it is the before target attrition rate; for SSM, it is used as a switch--if > 0, then available; otherwise assumed not available.)
Rate#2 for day 1	21	25	Value of first rate#2. (NOTE: Rate#2 has different meanings for each threat type: for FIGHTER and BOMBER, it is an attrition rate; for SOF, it is the post-attack loss rate; for SSM, it is usage and might be a function of launchers or doctrine.)
Change day 2	26	30	(see change day explanation above)
Rate#1 for day 2	31	35	(see rate#1 explanation above)
Rate#2 for day 2	36	40	(see rate#2 explanation above)
Change day 3	41	45	(see change day explanation above)
Rate#1 for day 3	46	50	(see rate#1 explanation above)
Rate#2 for day 3	51	55	(see rate#2 explanation above)

Figure A-5. DEFINITIONS OF THREAT FILE INPUT (RECORD 2)

c. **Target File.** This file contains all the targets that will be considered in the scenario. At present, the target priority is established preemptively by the ordering in the file. The attacker will try to "saturate" (i.e., meet the primary attack quota) target (n) before beginning to attack target ($n+1$).

(1) *Format.* An example of the Target file and the file format is found in **Figure A-6** below.

```
Example:  Target File

    NUCAF #1               U     NUCAF            200000N   0600000E
    NUCSTOR #2             U     NUCSTOR          200000N   0600000E
    COB1                   U     COB              200000N   0600000E
    COB2                   U     COB              200000N   0600000E
    Alpha City             U *   SMPORT           200000N   0600000E
    Metropolis             U     LGPORT           200000N   0600000E
    COB3                   U     COB              200000N   0600000E
    COMMO1                 U     TEL FX           200000N   0600000E
    COMMO2 (XX Corps)      U     TEL FX           200000N   0600000E
    MSR #1 (grid a)        T *   HWY              200000N   0600000E
    MSR #1 (grid b)        T *   PORT             203000N   0630000E
    AMMO Site 1            A     AMMO             200000N   0600000E
    PORT3                  A     PORT             200000N   0600000E

Format:  Target File

             1         2         3         4         5         6         7         8
    12345678901234567890123456789012345678901234567890123456789012345678901234567890
         |                   | |     |                 |         |
    Target Name              | |  Installation        Lat       Long
                             | |     Code
              Nation/group --┘ |
                               └-- Saturation Flag
```

Figure A-6. TARGET FILE EXAMPLE

(2) *Explanation.* **Figure A-7** contains the definitions of the Target file input. One consideration to keep in mind is the location. The model does not know if the latitude and longitude are correct, but it assumes they are. The user should ensure that coordinate entries fall within known north-south, east-west ranges. While this might be somewhat tedious for a large number of targets, it is necessary because DAMOC accepts "out-of-range" conditions, whether real or inadvertent.

Input Item	Start Col	End Col	Description
Target Name	1	20	Full name of target installation
Nation/group	25	25	Installation group identification. Could be nationality ('U' = United States); could be organizational ('4' = 4th ASG or '7' = VII Corps); or it could identify foreign bases ('T' = Turkish, but 't' = United States in Turkey). The group code is presently used only to select report scope.
Saturation Flag	27	27	This overrides the primary attack axiom. Normally when an installation receives its primary attack quota, it is no longer attacked (except for suppression and reconstitution situations). For some targets (roads, railroads, pipelines) this is unrealistic. By setting this flag to '*' a target will continue to be hit by each successive threat group.
Installation Code	30	39	This code indicates to which class of notional installations this target belongs.
Latitude	50	56	Latitude of target in ddmmssH
Longitude	60	67	Longitude of target in dddmmssV

Figure A-7. DEFINITION OF TARGET FILE INPUT

d. **Damage Profile File.** The Damage Allocation Model does not directly calculate damage. It actually apportions attackers according to target priority and nominal sortie requirements necessary to achieve predetermined damage levels. Damage is derived from the damage profiles developed during the first phase of the methodology. In its studies using DAMOC, ESC has relied on the Navy Air Attack Simulation Program (NAASP) as the detailed damage model. The Damage Profile file represents the information extracted from the NAASP[2].

[2] There are several models that could conceivably be used: NAASP, AAP, TSARINA. ESC opted for the NAASP because of its PC-availability. What DAMOC needs from a damage model such as NAASP is a damage template which relates damage to an appropriate level of standardized attacks.

(1) *Format.* **Figure A-8** below is an example of a Damage Profile file and the file formats.

```
Example:  Damage Profile File

*NCAF                    FIGHTER           30    12                               <—type 1 record
  RUNWAY                 111A      11200    0.00      15.14        6.5            <—type 2 record
  TAXIWAY                112A      22400    0.00      30.08       14.38
  APRON                  113A    5490000    0.00       6.82        0
  RAILROAD               860A      12000    0.00       3.16        0
  10K POL TANKS (BBL)    411E      50000   27.20       4.88        0
  AFCT MAINT FAC         211F     450000  100.00      17.16        0
  WTR STOR FAC  (GAL)    841C     100000  100.00       3.72        0

Format:  Damage Profile File [record type 1]

          1         2         3         4         5         6         7         8
 12345678901234567890123456789012345678901234567890123456789012345678901234567890
 |    |                      |          |   |
 |  Installation Code      Threat       |   └─ Suppression Attacks
 |  (Notional Type)        Type         └───── Primary Attacks
 |
 └─if = '*' then new Profile beginning

Format:  Damage Profile File [record type 2]

          1         2         3         4         5         6         7         8
 12345678901234567890123456789012345678901234567890123456789012345678901234567890
          |            |       |         |         |          |
  Facility Description    JCS     Amount       %        Hits/     Critical
                        Catcode   onhand     damage    craters    craters
```

Figure A-8. DAMAGE PROFILE FILE EXAMPLE

(2) *Explanation.* An important element of a damage profile is the facility set. It is important that there be no misconception about the facility entries. They need not represent all the facilities presumed to exist at a notional installation, only the ones that are damaged in a postulated attack. Undamaged facilities could be added, but would be gratuitous. Therefore, the minimum facility list for a notional installation class contains all facility types that have been damaged. These facilities need not be the same for different attackers. One would not expect that a SPETSNAZ team would have the same targets as a FIGHTER bomber. The program retains dissimilar facility lists within a notional installation. There is an internal check--if there are multiple profiles (i.e., more than one attacker type for a notional installation), then the onhand amounts for common facilities must be the same. Internally, DAMOC uses percentages. It combines percentages of threat types. Thus, it makes a difference if a FIGHTER's 50% damage is for 10,000 sq. ft. of 211F maintenance facilities, while an SOF's 30% damage is for only 500 sq. ft. of the same facility. (NOTE: When the model identifies such inconsistencies, it does not completely reject the information. It does presume, however, that the first encountered amount is true. The user, therefore, is advised to review the "DAMAGE TEMPLATE" report where such inconsistencies are reported.) Definitions of Damage Profile file input for Records 1 and 2 are found in **Figures A-9 and A-10** respectively.

Input Item	Start Col	End Col	Description
Record flag	1	1	Since there can be a variable number of associated facility records in a profile set, a '*' in column 1 identifies the record as the start of a new installation-threat system profile (e.g., "COB").
Installation code	2	11	The code used to identify the class of notional installations
Threat type	26	35	Indicates which threat asset class (FIGHTER, etc.) profile is being defined for the notional class
Primary attacks	42	45	The number of attacks necessary to achieve threshold damage levels (derived from detailed damage model results)
Suppression attacks	47	50	For those installation classes that have runways and piers, suppression attacks can be designated. These are the portion of the primary attacks directed against the specific facilities.

Figure A-9. DEFINITIONS OF DAMAGE PROFILE FILE INPUT (RECORD 1)

Input Item	Start Col	End Col	Description
Facility name	13	22	A descriptive name of the facility in the profile
JCS Catcode	26	29	The code associated with the facility
Amount onhand	31	40	The total amount (sq ft, bbls, lf, etc.) of this facility class subject to damage
Percent damaged	45	50	The average % of the onhand amount that is damaged when a full primary attack is made on the notional installation
Hits/craters	45	50	The average hits or craters on the onhand facilities resulting from a primary attack
Critical craters	64	70	The average craters on runways (or taxiways in some cases) that must be repaired to attain a minimum operational strip as defined for the detailed damage model

Figure A-10. DEFINITIONS OF DAMAGE PROFILE FILE INPUT (RECORD 2)

e. **Run Control File.** The normal execution of the Damage Model begins with the user responding to questions. The responses set some key variables defining scenario and model parameters. While not onerous, there may be reasons why one would prefer to have the model obtain this information from a file rather than through keyboard entry. Under DOS this is easily done using redirection: **DAMOC.EXE < RunCntrl.Fyl.** By the same token, redirection can be used to direct output from the screen (the default) to a designated file: **DAMOC.EXE > Output.Fyl.** Moreover, the operations can be concatenated: **DAMOC.EXE < RunCntrl.Fyl > Output.Fyl.** This section portrays the form and contents of this alternative input.

(1) *Format.* **Figure A-11** below is an example of a Run Control file.

```
Example:  Run Control File

 30                  number of days in scenario
 ABC                 countries in reports (summary and installation)*
 5                   reports every 5 days
 5   3               double sortie & suppression periods
 4   5               SOF & SSM sortie periods
 10  1               reconstitution: cycle days & times to do it
 bases.t                   enemy bases file name
 threat.t                  the threat file
 damtable                  notional damage arrays
 targets.t                 installation list
 _________________

 * A country code (a single alphanumeric character) is user defined.  There is one
   requirement--the code must correspond to the convention used in the Target
   file.  Also, there are two reserved characters:  an "*" indicates that all
   countries are to be reported; an "!" indicates that installation reports are
   not required.
```

Figure A-11. RUN CONTROL FILE EXAMPLE

(2) *Explanation.* TURBO PASCAL allows numerical free formatting from the input device, with a space(s) acting as a delimiter. Textual input must, however, be confined to a specified field and length. The example in Figure A-11 above shows the number of data and file.name entries that are expected. The comments that appear to the right are ignored. The entries in this file define global variables and identify appropriate data files. Comments regarding each entry appear below.

(a) *Scenario Days.* This entry defines the number of days in the scenario (limitation: days ≤ 180).

(b) *Country Reports.* This entry defines which target damage information will be portrayed in rollup and installation reports (see Annex B). By choosing different subsets of country/group identifiers, reports are limited to only those qualifying installations. However, the report mask has no affect on the simulation itself. The model does not attack targets based on their country/group ("*" = report all installations; "!" = summary reports only, no installations).

(c) *Report Frequency.* The summary reports are produced at set intervals, or cycles. If, in a 50-day scenario, one wanted summary reports every other day for the first 10 days and every 10th day thereafter, one would run the model twice: the first run would set report frequency to 2 days and scenario length to 10 days; the second would set frequency to 10 days and length to 50 days. The results will be consistent because the reports cycles have nothing to do with targeting (limitation: number of summary report periods ≤ 10).

(d) *Double Sorties.* The model will permit double sorties for aircraft during the first days of the scenario. This is to simulate the likely "surge" capability of the attackers during that period. This entry indicates the number of "surge" days (limitation: none, days = 0, 1, 2, 3, ...).

(e) *Suppression Periods.* Suppression attacks are defined for certain installation damage profiles. Their intent is to re-target certain facility types (pavement and piers) that can be repaired, thus restoring, to some degree, installation operability. This entry determines how many days after receiving its primary quota an installation can expect suppression attacks. Such attacks will continue until the suppression quota is reached, at which time the installation's suppression clock will be reset.

(f) *SOF and SSM Periods.* Since SOF and SSM assets are limited to some degree by delivery means, the user can husband these assets by setting use cycles. For example, the user might indicate that SOF will only be used every 4th day and SSM will only be used every 5th day.

(g) *Reconstitution.* One feature in the model allows a user to reset all damage counters. This theoretically simulates repair. Currently, it can only be done globally--damage is reset at all installations at the same time. If reconstitution is not wanted, simply set the cycle to "length-of-scenario + 1."

(h) *File Names.* Self explanatory.

4. **CHANGES.** The damage methodology has been used by ESC in three assessment studies. The first study created the requirement for DAMOC's existence; the second study identified other desirable features to be added to the model (e.g., ranging); the third study recognized the desirability of combining notional and actual target profiles. The formats and data defined in this annex reflect current needs. From experience, however, one might anticipate that the model and ESC's overall damage methodology will continue to evolve. This will doubtlessly require associated changes to input and formats.

Blank Page

LAST PAGE OF ANNEX A

ANNEX B

OUTPUT DESCRIPTIONS

ANNEX B

OUTPUT DESCRIPTIONS

Paragraph **Page**

Figure

1. **PURPOSE.** This annex provides examples of the various reports produced by the Damage Allocation Model (DAMOC).

2. **SCOPE.** Reports and other routine information described in this annex represent the output produced by DAMOC. A review of this section will assist the user in interpreting output from DAMOC. As similarly stated in the Scope of Annex A, however, no attempt is made here to describe output from any damage model used in conjunction with DAMOC.

3. **OUTPUT DESCRIPTIONS.** Output in DAMOC can be categorized into three areas: input verification, results, and execution monitoring.

a. **Input Verification.** With several interrelated input files, DAMOC attempts to assure data consistency to the extent possible. Since the user is given the freedom to define several fields (although standardized codes are desirable), the model is relegated to resolving differences or omissions rather than judging correctness. It can't know when the user meant to do something different. The user should, therefore, review the input verification section of the output. This is especially important since the normal course is to reject questionable entries, but not necessarily to terminate processing. Because the program does not consider inconsistencies to be fatal errors, execution and results might look correct, even though threat or target data may have been omitted. The various reports described below can assist the user in verifying content.

(1) *Scenario Queries.* When DAMOC is executed interactively (the default mode), the program asks a series of questions to set various scenario parameters and identify data files to be used (See **Figure B-1**). For a fuller explanation of the desired responses, the user is referred to Annex A's discussion of the Run Control file (the alternative to interactive processing).

```
Enter the number of days in the scenario ->
Select country codes ( a "*" means all included)-->
Length of period (and report cycle) -->
Enter days of double sorties & suppression period-->
enter SOF & SSM frequencies ->
enter reconstitution period and number-->
enter filename of threat bases-->
enter filename of threat systems-->
enter filename of installation-attack profiles-->
enter filename of target installations -->
```

Figure B-1. SCENARIO DEFINITION

(2) ***Threat Bases.*** The first data to be read in is the Threat Base file. This establishes the air bases and other military installations where threat assets can be located. **Figure B-2** shows an example of the list of bases produced by DAMOC. The base code is in brackets and the *x* and *y* coordinates (originally expressed in latitude and longitude) is expressed in radians.

```
.......attack bases defined

    airbase number 1      [base0]      0.17453    0.87266
    airbase number 5      [base5]      0.26180    1.04720
    airbase #2            [base2]      0.19199    0.90757
    airbase sac           [base3]      0.20944    0.94248
    corps hq              [base4]      0.22689    0.98000
    fixed launch #1       [base6]      0.24435    1.02684
    airbase forward       [base1]      0.26180    1.01287
```

Figure B-2. THREAT BASES SUMMARY

(3) ***Threat Systems.*** When processing the Threat Systems file, two fields are checked: the *threat type* (must be one of the four defined classes) and the *threat base* (must be a base code defined in the Threat Bases file). In the example shown in **Figure B-3**, an entry is rejected because DAMOC could not find one of the bases. After processing the data, a list of accepted systems, along with their type and range, is reported.

```
- redeploy error in BADGERS           BOMBER   base3  750   20   10   3base9
..threat rejected --BADGERS           BOMBER   base3  750   20   10   3base9

 .......threat definition

          FLOGGERS     [FIGHTER   ]  1281.0
          FENCERS      [FIGHTER   ]   500.0
          SPETSNAZ DIV [SOF       ]   150.0
          SPETSNAZ COR [SOF       ]   300.0
          SCUD         [SSM       ]   400.0
```

Figure B-3. THREAT SYSTEMS SUMMARY

(4) ***Damage Profiles.*** The damage profiles comprise the largest input file. While the file is being processed, information is printed, followed by two reports that summarize facility and profile-related information.

(a) *Installation Log.* Each time a new notional or actual installation is encountered in the file, a message records its creation. **Figure B-4** gives an example of that record.

```
instalprofile created for [NCAF      ]
instalprofile created for [MOB       ]
instalprofile created for [COB       ]
instalprofile created for [FLDCMP    ]
instalprofile created for [TELFX     ]
instalprofile created for [TELFD     ]
instalprofile created for [HAWK      ]
instalprofile created for [LGPORT    ]
instalprofile created for [SMPORT    ]
instalprofile created for [LGPOL     ]
instalprofile created for [SMPOL     ]
instalprofile created for [NATOPL    ]
instalprofile created for [POLFLD    ]
instalprofile created for [AMMOFX    ]
instalprofile created for [AMMOFD    ]
instalprofile created for [STORFX    ]
instalprofile created for [STORFD    ]
instalprofile created for [CPOWER    ]
instalprofile created for [FPOWER    ]
instalprofile created for [WATER     ]
instalprofile created for [WHIWAY    ]
instalprofile created for [NHIWAY    ]
instalprofile created for [HWYBRG    ]
instalprofile created for [HWYTNL    ]
instalprofile created for [RR        ]
instalprofile created for [RRBRG     ]
instalprofile created for [RRTNL     ]
instalprofile created for [RRYD      ]
```

Figure B-4. NOTIONAL INSTALLATION CREATION RECORD

(b) *Facility List.* Rather than redundantly retaining the full text name of each facility within a profile, DAMOC creates a facility master list which is indexed by the catcode. During the processing of a profile, each facility entry is checked against the master list. If the facility is not found, a new entry (facility name and catcode) is placed in the master list. **Figure B-5** shows the contents of the master list that was created after processing one version of installation profiles. Ranking is by JCS catcode. Note that this list determines the entries and order in the summary rollup reports.

```
#REFERENCE JCS CATCODE LIST:
     (     1)  111A----RUNWAY
     (     2)  111C----HELO LANDING PAD
     (     3)  112A----TAXIWAY
     (     4)  113A----APRON
     (     5)  123A----POL DISPENSING PT
     (     6)  125A----POL PIPELINE
     (     7)  125B----VALVE BOX     (EA)
     (     8)  131A----ANTENNA       (EA)
     (     9)  131B----COMMO EQUIP FLD
     (    10)  131D----TRANSMITTER BLDG
     (    11)  131E----TELEMETRY BLDG
     (    12)  133A----CONTROL TOWER
     (    13)  141D----AIRCRAFT SHLTR
     (    14)  151C----PIER
     (    15)  159C----WATER FRONT OPS
     (    16)  163A----LANDING DOCK
     (    17)  211F----AFCT MAINT FAC
     (    18)  216A----AMMO MAINT FAC
     (    19)  217A----COMMO MAINT FAC
     (    20)  219A----FAC MAINT SHOP
     (    21)  411A----FUEL TANK     (BBL)
     (    22)  411B----POL BLADDERS  (BBL)
     (    23)  411D----3K POL TANKS  (BBL)
     (    24)  411E----10K POL TANKS (BBL)
     (    25)  411F----POL STOR YARD
     (    26)  421A----AMMO STORAGE FAC
     (    27)  422A----NUC AMMO STOR
     (    28)  425A----OPEN AMMO STORAGE
     (    29)  441A----WAREHOUSE (PORT)
     (    30)  442A----GP COVERED STOR
     (    31)  451A----GP OPEN STORAGE
     (    32)  452A----PORT OPEN STORAGE
     (    33)  610A----ADMIN FACILITY
     (    34)  811A----ELECT SUB-STAT
     (    35)  841C----WTR STOR FAC  (GAL)
     (    36)  842A----PUMP UNIT     (EA)
     (    37)  842B----WATER PIPELINE
     (    38)  851A----HIGHWAY WIDE
     (    39)  851B----TWO LANE ROAD
     (    40)  851C----ROAD BRIDGE    (SPANS)
     (    41)  860A----RAILROAD
     (    42)  860B----RAILROAD BRIDGE (SPANS)
     (    43)  860C----RAILROAD TUNNEL
     (    44)  860D----RAILROAD YARD
     (    45)  9999----FLD CMD POST
     (    46)  999B----LOADER/TRANSPTR
     (    47)  999C----CRANE         (EA)
     (    48)  999D----FLD CMD POST
     (    49)  999F----GENERATORS    (KW)
```

Figure B-5. FACILITY/CATCODE MASTER LIST

(c) *Damage Profile Summary.* The final subreport for damage profiles is the template summary. It summarizes individual installation threat damage profiles. The two numbers that appear in brackets after the threat name give the primary and suppression attack quotas. That, in turn, is followed by the damaged facility count for the installation threat. During the production of this summary, on-hand facility amounts are checked across profiles within a notional installation. The example (**Figure B-6**) shows an inconsistency message for ammo storage (421A) at a MOB. This tells the user that the on-hand conformity requirement was violated. DAMOC builds one facility list for each notional installation and only one on-hand amount is kept. If on-hand assets are not equal, it makes a large difference when calculating damage. For example, if FIGHTERS and SOF each damage 50% of on-hand assets, but the SOF's presumed target was two orders of magnitude smaller than the planes, then adding the percents together will result in 100% damage when the SOF contribution should have been 0.5% of the larger on-hand figure. The program presumes the first on-hand amount is correct. Upon encountering such a message, the user must resolve the discrepancy. Note also that in this example there is no profile for SOF attacks against nuclear capable airfields (NCAFs). This may indicate a deliberate policy (without a profile, DAMOC will not target SOF against NCAFs), or perhaps something was inadvertently left out of the file or mislabeled.

```
#==DAMAGE TEMPLATE INPUT SUMMARY!

NCAF
    FIGHTER     [  30   12]       13  facilities damaged.
    BOMBER      [  24   12]       13  facilities damaged.
    SSM         [   1    0]        2  facilities damaged.
MOB
    FIGHTER     [  30   12]       13  facilities damaged.
    BOMBER      [  24   12]       13  facilities damaged.
    SOF         [   1    0]        1  facilities damaged.
    SSM         [   1    0]        2  facilities damaged.
* on-hand inconsistency for 421A--[   1200 <>     120000]   <----Inconsistency Error
COB
    FIGHTER     [  14   12]        9  facilities damaged.
    BOMBER      [  14   12]        6  facilities damaged.
    SOF         [   1    0]        1  facilities damaged.
    SSM         [   1    0]        1  facilities damaged.
FLDCMP
    FIGHTER     [   1    0]        1  facilities damaged.
    BOMBER      [   1    0]        1  facilities damaged.
    SOF         [   1    0]        1  facilities damaged.
    SSM         [   1    0]        1  facilities damaged.
TELFX
    FIGHTER     [   4    0]        5  facilities damaged.
    BOMBER      [   2    0]        5  facilities damaged.
    SOF         [   1    0]        2  facilities damaged.
    SSM         [   1    0]        5  facilities damaged.
TELFD
    FIGHTER     [   1    0]        1  facilities damaged.
    BOMBER      [   1    0]        1  facilities damaged.
    SSM         [   1    0]        1  facilities damaged.
HAWK
    FIGHTER     [   1    0]        5  facilities damaged.
    BOMBER      [   1    0]        5  facilities damaged.
    SOF         [   1    0]        3  facilities damaged.
    SSM         [   1    0]        5  facilities damaged.
LGPORT
    FIGHTER     [  18   14]        4  facilities damaged.
    BOMBER      [  18   14]        3  facilities damaged.
    SSM         [   1    0]        3  facilities damaged.
                          .
                          .
```

Figure B-6. DAMAGE PROFILE SUMMARY

(5) *Targets*. The target file is the list of installations in priority order. These real-world locations also designate to which country/group and which notional installation they belong. The final input is latitude and longitude. **Figure B-7** gives an example of the output that accompanies the target file processing. It shows several targets being rejected because they do not have recognizable notional installation designations. DAMOC has no way of knowing if the country/group is right or wrong since it is user defined. The user, therefore, should check this field (Nat=?), especially if installation subset reporting will be used.

```
..no ref-install for NUCAF    a      U    NUCAF         200C00N   0600000E
..no ref-install for NUCSTOR  b      U    NUCSTOR       200000N   0600000E
..no ref-install for COMMO1   k      U    TEL FX        200000N   0600000E
..no ref-install for COMMO2   l      U    TEL FX        200000N   0600000E
..no ref-install for PORT2    aa     T    PORT          200000N   0600000E
..no ref-install for AMMO     bb     A    AMMO          200000N   0600000E
..no ref-install for PORT3    cc     A    PORT          200000N   0600000E


#Regional installations in priority order
(    1)        COB1    c       [Nat=U]      COB
(    2)        COB2    d       [Nat=U]      COB
(    3)        xyz             [Nat=U]      SMPORT
(    4)        ABC             [Nat=U]      LGPORT
(    5)        COB3    e       [Nat=U]      COB
(    6)        COB4    f       [Nat=U]      COB
(    7)        COB5    g       [Nat=U]      COB
(    8)        COB6    h       [Nat=T]      COB
(    9)        COB7    i       [Nat=T]      COB
(   10)        COB8    j       [Nat=T]      COB
```

Figure B-7. TARGET INSTALLATION SUMMARY

b. **Results.** The reason for DAMOC's existence is a need for a reasonable estimate of war damage at echelons above corps. When one considers the scores or hundreds of targets, the varying number o days or periods, the groups of targets, and the varying target makeups, it is understandable that no single report can capture all information.

(1) ***Facility Rollups.***

(a) *Period Reports.* Among the scenario parameters are report frequency and country codes. While the frequency has no influence on damage calculations, it does set the timing of damage result summaries. These reports give periodic rollups of facility damage across a subset of installations defined by the country codes (**Figure B-8**). It is important to recognize that the on-hand and damage amounts in the report are only for targets of the countries or organizations in this subset.

#Summary Report for period ending day 5 for countries T

CatCode	Facility	On-hand	Damage	Hits	Craters
[111A]	RUNWAY	67200	0.0	211.01	19.17
[111C]	HELO LANDING PAD	0	0.0	0.00	0.00
[112A]	TAXIWAY	23400	0.0	81.68	0.00
[113A]	APRON	1500000	0.0	10.91	0.00
[123A]	POL DISPENSING PT	0	0.0	0.00	0.00
[125A]	POL PIPELINE	0	0.0	0.00	0.00
[125B]	VALVE BOX (EA)	0	0.0	0.00	0.00
[131A]	ANTENNA (EA)	0	0.0	0.00	0.00
[131B]	COMMO EQUIP FLD	0	0.0	0.00	0.00
[131D]	TRANSMITTER BLDG	0	0.0	0.00	0.00
[131E]	TELEMETRY BLDG	0	0.0	0.00	0.00
[133A]	CONTROL TOWER	16875	0.0	0.12	0.00
[141D]	AIRCRAFT SHLTR	195000	1950.0	12.00	0.00
[151C]	PIER	0	0.0	0.00	0.00
[159C]	WATER FRONT OPS	0	0.0	0.00	0.00
[163A]	LANDING DOCK	0	0.0	0.00	0.00
[211F]	AFCT MAINT FAC	180000	0.0	0.00	0.00
[216A]	AMMO MAINT FAC	0	0.0	0.00	0.00
[217A]	COMMO MAINT FAC	0	0.0	0.00	0.00
[219A]	FAC MAINT SHOP	0	0.0	0.00	0.00
[411A]	FUEL TANK (BBL)	0	0.0	0.00	0.00
[411B]	POL BLADDERS (BBL)	0	0.0	0.00	0.00
[411D]	3K POL TANKS (BBL)	0	0.0	0.00	0.00
[411E]	10K POL TANKS (BBL)	0	0.0	0.00	0.00
[411F]	POL STOR YARD	75000	18750.0	11.22	0.00
[421A]	AMMO STORAGE FAC	0	0.0	0.00	0.00
[425A]	OPEN AMMO STORAGE	360000	25272.0	11.40	0.00
[441A]	WAREHOUSE (PORT)	0	0.0	0.00	0.00
[442A]	GP COVERED STOR	0	0.0	0.00	0.00
[451A]	GP OPEN STORAGE	0	0.0	0.00	0.00
[452A]	PORT OPEN STORAGE	0	0.0	0.00	0.00
[610A]	ADMIN FACILITY	0	0.0	0.00	0.00
[811A]	ELECT SUB-STAT	45000	0.0	0.00	0.00
[841C]	WTR STOR FAC (GAL)	0	0.0	0.00	0.00
[842A]	PUMP UNIT (FA)	0	0.0	0.00	0.00
[851A]	HIGHWAY WIDE	0	0.0	0.00	0.00
[851B]	TWO LANE ROAD	0	0.0	0.00	0.00
[851C]	ROAD BRIDGE (SPANS)	0	0.0	0.00	0.00
[860A]	RAILROAD	0	0.0	0.00	0.00
[860B]	RAILROAD BRIDGE (SPANS)	0	0.0	0.00	0.00
[860C]	RAILROAD TUNNEL	0	0.0	0.00	0.00
[860D]	RAILROAD YARD	0	0.0	0.00	0.00
[999?]	FLD CMD POST	0	0.0	0.00	0.00
[999B]	LOADER/TRANSPTR	0	0.0	0.00	0.00
[999C]	CRANE (EA)	0	0.0	0.00	0.00
[999D]	FLD CMD POST	0	0.0	0.00	0.00
[999F]	GENERATORS (KW)	0	0.0	0.00	0.00

Figure B-8. FACILITY DAMAGE ROLLUP BY PERIOD

(b) *Scenario Rollup.* This report is identical to the period report except that it is for the entire scenario. Format and interpretation are the same.

(2) *Installation Summaries.*

(a) *Attack Record.* To see the pattern of which, or to know when, installations were actually attacked, a record of attacks is kept for each installation. **Figure B-9** gives an example of this information which is routinely printed along with the facility damage results for each installation. It indicates the number of sorties (attacks), by threat type, by period. For air

attacks, it also indicates primary ("/p") and suppression ("/s") sorties. In this figure, there are two instances of the period report. This is because an attack record is created at the beginning of the scenario and at the time of reconstitution of an installation. Here a reconstitution necessarily occurred sometime in periods 2, 3, or 4.

		PERIODS					
		1	2	3	4	5	6
[FIGHTER	/p]	14.0	0.0	0.0	0.0	0.0	0.0
[BOMBER	/p]	0.0	0.0	0.0	0.0	0.0	0.0
[SOF	/p]	0.0	0.0	0.0	0.0	0.0	0.0
[SSM	/p]	0.0	0.0	0.0	0.0	0.0	0.0
[FIGHTER	/s]	12.0	11.0	0.0	0.0	0.0	0.0
[BOMBER	/s]	0.0	1.0	0.0	0.0	0.0	0.0
[FIGHTER	/p]	0.0	0.0	0.0	7.9	4.2	0.0
[BOMBER	/p]	0.0	0.0	0.0	1.8	0.1	0.0
[SOF	/p]	0.0	0.0	0.0	0.0	0.0	0.0
[SSM	/p]	0.0	0.0	0.0	1.0	0.0	0.0
[FIGHTER	/s]	0.0	0.0	0.0	0.0	0.0	4.2
[BOMBER	/s]	0.0	0.0	0.0	0.0	0.0	1.8

Figure B-9. INSTALLATION ATTACK SUMMARY BY PERIOD

(b) *Facility Damage*. The second part of the Installation Summary shows facility damage (**Figure B-10**). Unlike the facility summaries that list <u>all</u> defined catcodes, only the facilities actually included on the installation are listed for damage and hits. The critical crater section is limited to appropriate facilities (pavement and piers).

FACILITY DAMAGE

Facility	Catcode	period 1	period 2	period 3	period 4	period 5	period 6
RUNWAY	111A	0.0	0.0	0.0	0.0	0.0	0.0
TAXIWAY	112A	0.0	0.0	0.0	0.0	0.0	0.0
APRON	113A	0.0	0.0	0.0	39600.0	3027.2	0.0
CONTROL TOWER	133A	0.0	0.0	0.0	46.3	3.5	0.0
AIRCRAFT SHLTR	141D	0.0	0.0	0.0	650.0	0.0	0.0
AFCT MAINT FAC	211F	0.0	0.0	0.0	0.0	0.0	0.0
POL STOR YARD	411F	6250.0	0.0	0.0	5112.3	1997.5	0.0
OPEN AMMO STORAGE	425A	8424.0	0.0	0.0	4730.9	2527.2	0.0
ELECT SUB-STAT	811A	0.0	0.0	0.0	0.0	0.0	0.0

FACILITY HITS

Facility	Catcode	period 1	period 2	period 3	period 4	period 5	period 6
RUNWAY	111A	53.0	27.1	0.0	19.2	8.3	14.3
TAXIWAY	112A	20.5	10.5	0.0	7.4	3.2	5.5
APRON	113A	2.3	1.3	0 0	1.0	0.4	0.8
CONTROL TOWER	133A	0.0	0.0	0.0	0.1	0.0	0.0
AIRCRAFT SHLTR	141D	0.0	0.0	0.0	4.0	0.0	0.0
AFCT MAINT FAC	211F	0.0	0.0	0.0	0.0	0.0	0.0
POL STOR YARD	411F	3.7	0.0	0.0	3.4	1.2	0.0
OPEN AMMO STORAGE	425A	3.8	0.0	0.0	3.6	1.3	0.0
ELECT SUB-STAT	811A	0.0	0.0	0.0	0.0	0.0	0.0

CRITICAL CRATERS

RUNWAY	111A	5.9	2.7	0.0	1.7	0.9	1.1
TAXIWAY	112A	0.0	0.0	0.0	0.0	0.0	0.0
APRON	113A	0.0	0.0	0.0	0.0	0.0	0.0

Figure B-10. INSTALLATION FACILITY DAMAGE SUMMARY BY PERIOD

(3) *Installation Class Attacks*. It is sometimes useful to check how the installation classes are being covered by attacks. **Figure B-11** shows an example of accumulated sorties sent against installation classes, by period. All sorties are added together (i.e., FIGHTER, BOMBER, SOF, and SSM). The user can use this report to appraise the impact of priority ordering, suppression, and reconstitution on attack allocations.

Sortie/Installation Breakout: ....

<< NCAF	>>	0.0	0.0	0.0	0.0	0.0	0.0
<< MOB	>>	0.0	0.0	0.0	0.0	0.0	0.0
<< COB	>>	78.0	19.7	0.0	3.0	0.0	0.0
<< FLDCMP	>>	0.0	0.0	0.0	0.0	0.0	0.0
<< TELFX	>>	0.0	0.0	0.0	0.0	0.0	0.0
<< TELFD	>>	0.0	0.0	0.0	0.0	0.0	0.0
<< HAWK	>>	0.0	0.0	0.0	0.0	0.0	0.0
<< LGPORT	>>	0.0	0.0	0.0	0.0	0.0	0.0
<< SMPORT	>>	0.0	0.0	0.0	0.0	0.0	0.0
<< LGPOL	>>	0.0	0.0	0.0	0.0	0.0	0.0
<< SMPOL	>>	0.0	0.0	0.0	0.0	0.0	0.0
<< NATOPL	>>	0.0	0.0	0.0	0.0	0.0	0.0
<< POLFLD	>>	0.0	0.0	0.0	0.0	0.0	0.0
<< AMMOFX	>>	0.0	0.0	0.0	0.0	0.0	0.0
<< AMMOFD	>>	0.0	0.0	0.0	0.0	0.0	0.0
<< STORFX	>>	0.0	0.0	0.0	0.0	0.0	0.0
<< STORFD	>>	0.0	0.0	0.0	0.0	0.0	0.0
<< CPOWER	>>	0.0	0.0	0.0	0.0	0.0	0.0
<< FPOWER	>>	0.0	0.0	0.0	0.0	0.0	0.0
<< WATER	>>	0.0	0.0	0.0	0.0	0.0	0.0
<< WHIWAY	>>	0.0	0.0	0.0	0.0	0.0	0.0
<< NHIWAY	>>	0.0	0.0	0.0	0.0	0.0	0.0
<< HWYBRG	>>	0.0	0.0	0.0	0.0	0.0	0.0
<< HWYTNL	>>	0.0	0.0	0.0	0.0	0.0	0.0
<< RR	>>	0.0	0.0	0.0	0.0	0.0	0.0
<< RRBRG	>>	0.0	0.0	0.0	0.0	0.0	0.0
<< RRTNL	>>	0.0	0.0	0.0	0.0	0.0	0.0
<< RRYD	>>	0.0	0.0	0.0	0.0	0.0	0.0

Figure B-11. INSTALLATION CLASS ATTACK SUMMARY BY PERIOD

(4) *Sortie Summaries.* The previous report rolled up total attacks against installation class. The sortie summary information records the total number of available sorties on a daily basis. **Figure B-12** shows the four threat types (e.g., 1 = FIGHTER, 2 = BOMBER) and the total available sorties each day for a 30-day scenario. Note that this report lists "available" sorties while the installation attack report lists "actual" sorties.

```
Sorties summary:...........

(T=1)154125101 82 66 | 30 29 28 28 22 | 22 22 22 22 22 |
      22 22 22 22 22 | 22 22 22 22 22 | 22 22 22 22 22 |
(T=2)  0  0  0  0 20 | 20 20 20 20 11 |  9  7  6  6  6 |
       6  6  6  6  6 |  6  6  6  6  6 |  6  6  6  6  6 |
(T=3)  0  0  0  0 17 |  0  0  0 12  0 |  0  0 27  0  0 |
       0 21  0  0  0 | 17  0  0  0 13 |  0  0  0 11  0 |
(T=4)  0  0  0  0  0 |  0  0  0  0  0 |  0  0  0  0  0 |
      10  0  0  0  0 |  8  0  0  0  0 |  6  0  0  0  0 |
```

Figure B-12. DAILY THREAT CLASS SORTIE SUMMARY

c. **Execution Monitoring.**

(1) *Machine Performance.* At different times during execution information is printed regarding memory and execution performance (**Figure B-13**). This information is useful in assuring that nothing untoward occurs. Heap memory readouts occur at the start and end of execution. If memory demands become a problem, the user might check to see if as much memory as possible has been made available. In the example, the initial heap measure is 427,728. Since the instruction portion of DAMOC occupies less than 40K, then *[640 - 40] - 430 = 170* implies that around 170K of memory is not being used by DAMOC. The user should remove unnecessary drivers and resident programs to free up some of this memory. The other message shown here indicates how much time has elapsed since the previous elapsed-time readout. This allows the user to ensure that execution time for various phases of the model is reasonable.

```
 heap memory =    427728, time is 16:21:45
                      .
                      .
=====day(1)  [elapsed time is   0.083 minutes]
=====day(2)  [elapsed time is   0.017 minutes]
=====day(3)  [elapsed time is   0.000 minutes]
=====day(4)  [elapsed time is   0.000 minutes]
=====day(5)  [elapsed time is   0.000 minutes]
                      .
                      .
 heap memory =    397600, time is 16:22:03
```

Figure B-13. MODEL PERFORMANCE AND TIMING MESSAGES

(2) *Event Log.* An event history is produced automatically during each execution of the model. It is written to file "SORTIE.LOG" (**Figure B-14**). Currently, this is not a user-designated file and a name change of the existing file would be required if one wanted to retain the information. The information found in the file is very useful to confirm that the attacks and movement are happening as they should. **Figure B-15** gives an annotated excerpt of a typical SORTIE.LOG file. The user should read these comments.

```
(  1)   1 /COB1      c          14.0-FLOGGERS      77.0
(  2)   1 /COB2      d          14.0-FLOGGERS      63.0
(  3)   1 /xyz                   8.0-FLOGGERS      49.0
(  4)   1 /ABC                  18.0-FLOGGERS      41.0
(  5)   1 /COB3      e          14.0-FLOGGERS      23.0
(  6)   1 /COB4      f          14.0-FLOGGERS       9.0
(  7)  ...[ 1]...FLOGGERS     - unused   0.0
(  8)   1 /xyz                   8.0-FENCERS       77.0
(  9)   1 /COB4      f           5.0-FENCERS       69.0
( 10)   1 /COB5      g          14.0-FENCERS       63.9
( 11)   1 /COB6      h          14.0-FENCERS       49.9
( 12)   1 /COB7      i          14.0-FENCERS       35.9
( 13)   1 /COB8      j          14.0-FENCERS       21.9
( 14)  ...[ 1]...FENCERS      - unused   7.9
( 15)  ...[ 1]...BADGERS      - unused   0.0
( 16)  ...[ 1]...SPETSNAZ DIV - unused   0.0
( 17)  ...[ 1]...SPETSNAZ COR - unused   0.0
( 18)  ...[ 1]...SCUD         - unused   0.0
( 19)              .                  .
( 20)   3 /xyz                   8.0-FLOGGERS      50.5
( 21)  ...[ 3]...FLOGGERS     - unused  42.5
( 22)   3 /xyz                   8.0-FENCERS       50.5
( 23)  ...[ 3]...FENCERS      - unused  42.5
( 24)  ---[ 3]--- BADGERS    moves from airbase sac   to airbase #2
( 25)              .                  .
( 26)   4 /COB1      c          12.0 (sup) FLOGGERS      40.9
( 27)   4 /COB2      d          12.0 (sup) FLOGGERS      28.9
( 28)   4 /xyz                   8.0-FLOGGERS      16.9
( 29)   4 /ABC                  14.0 (sup) FLOGGERS       8.9
( 30)  ...[ 4]...FLOGGERS     - unused   0.0
( 31)   4 /xyz                   8.0-FENCERS       40.9
( 32)   4 /ABC                   5.1 (sup) FENCERS       32.9
( 33)   4 /COB3      e          12.0 (sup) FENCERS       27.8
( 34)   4 /COB4      f          12.0 (sup) FENCERS       15.8
( 35)   4 /COB5      g          12.0 (sup) FENCERS        3.8
( 36)  ...[ 4]...FENCERS      - unused   0.0
( 37)  ...[ 4]...BADGERS      - unused   0.0
( 38)  ...[ 4]...SPETSNAZ DIV - unused   0.0
( 39)  ...[ 4]...SPETSNAZ COR - unused   0.0
( 40)              .                  .
( 41)  16 /COB1      c           1.0-SCUD          10.0
( 42)  16 /COB2      d           1.0-SCUD           9.0
( 43)  16 /xyz                   1.0-SCUD           8.0
( 44)  16 /ABC                   1.0-SCUD           7.0
( 45)              .                  .
( 46)              .                  .
( 47)  ...[17]...BADGERS      - unused   0.0
( 48)  ...[17]...SPETSNAZ DIV - unused  14.0
```

Figure B-14. SORTIE LOG FILE EXTRACT

Line	Explanation
1	On day 1 Target "COB1 c" requires 14 sorties to satisfy a primary FIGHTER attack quota of 14. There are 77 available FLOGGER sorties. Since 14 < 77 "COB1 c" can be saturated. An indicator in the target will indicate when primary sorties have been met, and FLOGGER availables are reduced accordingly.
6	On day 6 requirements at "COB4 f" exceed available FLOGGERS (14 > 9). Rather than look for other targets whose requirements are less than or equal to 9, the model allocates the 9 against the target.
8	Target "xyz" was saturated by FLOGGERS on day 1 (see line 3). "Xyz" has come up again as a potential target on day 1. The reason for this is that "xyz" must have had its saturation flag set. Therefore with each change of threat asset (in this case FLOGGER to FENCER) it can be targeted again as if it were unscathed by previous attacks.
9	Here target "COB4 f" is finished off. The requirement is now for 5 air sorties, and that is well within FENCER availabilities. Note that a partially saturated condition can continual indefinitely.
14	The model attempts to allocate threat assets completely. If all targets are saturated, out of range, or immune to attack (no damage profile for asset type), then the model has no place to put excess capability. The message on this line indicates the uncommitted assets. (Upon examination one would find that FENCERS have a shorter range than the previously assigned FLOGGERS. Perhaps reversing the order would better use assets.)
24	The message here records a redeployment. A BADGER has moved from one base to another.
26	The "(sup)" found in this entry indicates that this was a suppression attack. Indeed assuming "COB1 c" was saturated on day 1 (see line 1) and that the suppression cycle was set at 3, then this is as it should be. A check of the damage profiles would also verify that suppression attacks require 12, not 14, attacks for this notional installation class.
41	SCUDs are available on day 16 (cycle = 5) and are used against targets.
48	SOF are available but unused. Saturation is not the reason since no other SOF attacks had been made. The most likely reason is that targets are simply out of range.

Figure B-15. SORTIE LOG FILE EXTRACT--LINE EXPLANATIONS

Blank Page

LAST PAGE OF ANNEX B

ANNEX C

DAMOC DOCUMENTATION

ANNEX C

DAMOC DOCUMENTATION

1. **PURPOSE.** This annex is an overview of the structure and object types used in the Damage Allocation Model (DAMOC) Program.

2. **SCOPE.** DAMOC is written in TURBO PASCAL 5.5.[1] This version adds object-oriented programming (OOP) constructs to the popular PC programming language. ESC made good use of the new features throughout DAMOC. As a result, however, programmers familiar with PASCAL may have to disabuse themselves of certain preconceptions. Despite retaining all of the old commands, DAMOC's representation in PASCAL 5.5 is sufficiently different in style and approach to almost be a different language because of the OOP design. Appreciating these differences can take time, but is necessary to understand how these features are used. This annex is not a tutorial on 5.5 or OOP.[2] It addresses only DAMOC and consequently does not attempt to school the reader on virtual functions, constructors, inheritance, etc.

3. **LIMITATION.** ESC has not prepared line-by-line descriptions of the program units. Nor will ESC claim that the code is self-documenting--it isn't. On the other hand, if a user needs to change the program, it must be with knowledge of the program at code level. Inline code remarks are sometimes more misleading than helpful and frequently frustrate readability that indenting provides. One need only remember that after compilation and linking, the **code** is executed. While OOP is probably the best decomposition technique currently available, program changes should not be made in isolation. With the object-oriented approach used by DAMOC, someone looking at the code may have to reorient his/her programming paradigm in order to understand and to modify the code.

4. **OBJECT TYPE DESCRIPTIONS.** Becoming familiar with the object types used in DAMOC goes a long way toward understanding the structure of the model. Indeed, this is an often-quoted advantage of OOP-based systems. **Figure C-1** graphically represents the object type hierarchy and the variables and methods associated with the types. The individual object types (class definitions) are described below using a common framework (**Figures C-2 through C-17**). The name of the type and its ancestor, if applicable, are given first. Next the object variable attributes (TURBO PASCAL refers to them as fields) are defined. The variable types are also indicated. Arrays are indicated by a "[ ]" after the type. Finally, the functional and procedural attributes (TURBO PASCAL's methods) are described. The descriptions are intended to introduce the object types. The user must look to the actual code (in the appendices) to see how the concepts are realized. Some object methods have the same (e.g., "dump" or "build") or similar (e.g., "init" or "init2") names--this usually indicates a similar functional intent although perhaps with an entirely different code. For example, build methods are common to all set-type objects and internalize the creation of the disparate lists.

---

[1] *TURBO PASCAL 5.5· Object-Oriented Programming Guide* (Borland International, 1988).

[2] As OOP grows in popularity, scores of books are appearing on the subject. Not all of the books are equal, and some may even be misleading. One well known software pundit published a book about OOP using ADA, but only a short time later went on record saying one couldn't do OOP in ADA. While one can use OOP-like ideas such as decomposition and data localization in FORTRAN, C, ADA, etc., those languages lack the compiler-provided hallmarks of true object languages such as C^{++} and SIMULA (the nestor of OOP languages). Modelers with experience in other languages will go through a learning curve on the way to becoming object-oriented.

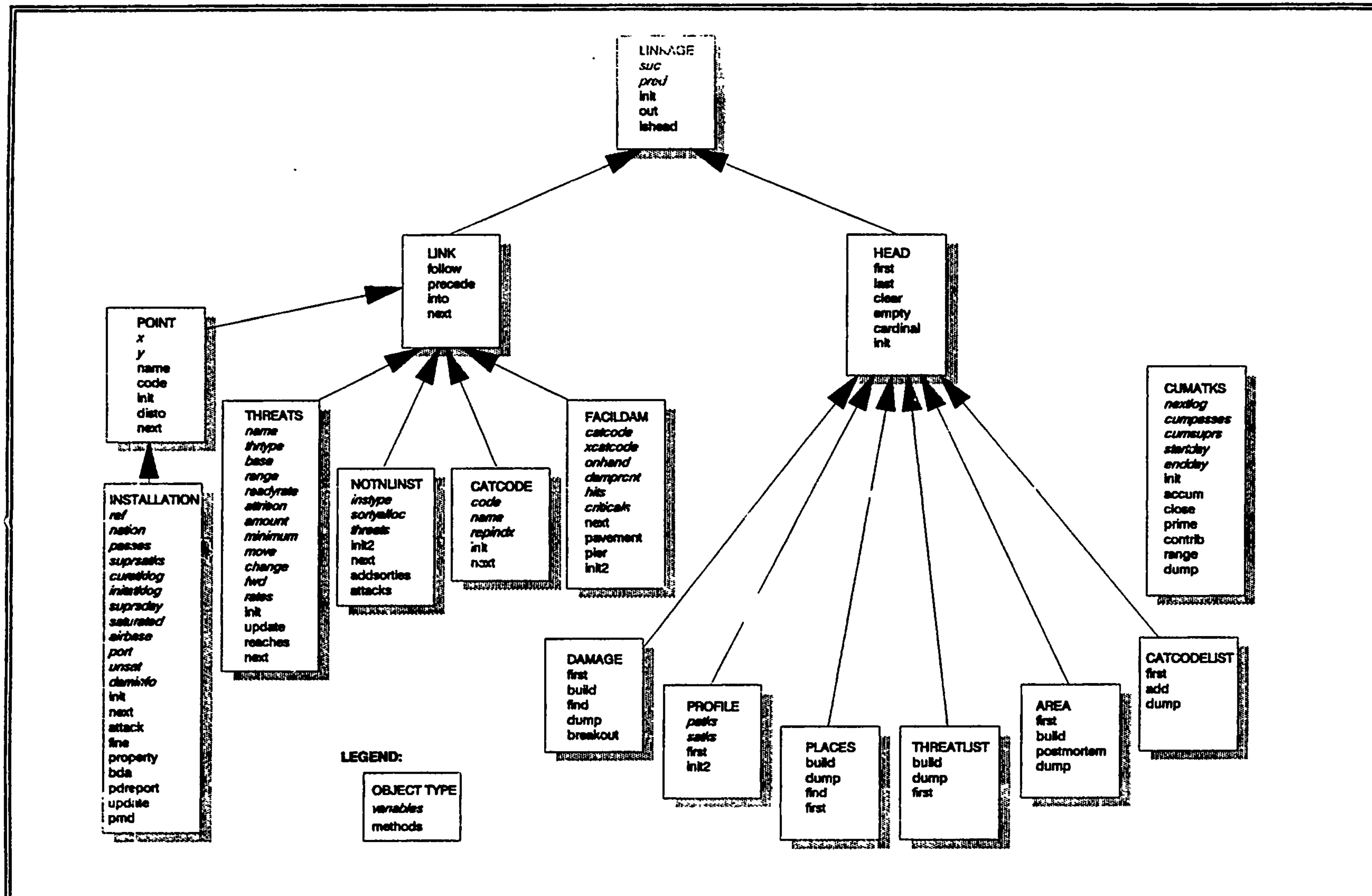

Figure C-1. DAMOC OBJECT TYPE CLASS HIERARCHY

a. **Linkage.** The main report discusses the purpose of the SIMSETx layer of the program, which contains facilities for the manipulation of circular two-way lists (a.k.a. *sets*). Attributes of object type *linkage* are not accessed directly. In a more recent version of PASCAL 5.5, they would be "protected" variables. *Link* and *head* are the derived types used by the modeler to construct his or her own two-way lists.

OBJECT TYPE: linkage		**ANCESTOR TYPE:** none
VARIABLES: suc pred	 reflinkage reflinkage	 points to the following linkage object points to the preceding linkage element
PROCEDURES: init out	 this is the PASCAL constructor called when creating a linkage object this extracts an object from a set, setting suc and pred equal to nil and adjusting linkage values of adjacent linkage objects	

Figure C-2. ATTRIBUTE DESCRIPTION OF OBJECT LINKAGE

b. **Link.** When an object is defined as a descendent of *link*, it enables list membership and can avail itself of the set manipulation routines. Various method attributes are provided to position the object upon insertion. Note that a *link* object can be in only one list at a time. If one wants to place a single object in several lists, then aliases (other *link* objects pointing to the common object) would have to be used.

OBJECT TYPE: link	**ANCESTOR TYPE:** linkage
PROCEDURES: follow precede into	 places object after a referenced linkage object places object before a referenced linkage object puts object into referenced list

Figure C-3. ATTRIBUTE DESCRIPTION OF OBJECT LINK

c. **Head.** Whereas *link* enables list or set membership, the list itself is established by instantiating objects or descendants of object type *head.*

OBJECT TYPE: head		ANCESTOR TYPE: linkage
FUNCTIONS:		
first	reflinkage	returns the first member of the set or nil if list is empty
last	reflinkage	returns the last member in the set
empty	boolean	true if set is empty, false otherwise
cardinal	integer	returns the number of objects in list
PROCEDURES:		
init	the constructor called when a head object is created	
clear	clears contents of a set	

Figure C-4. ATTRIBUTE DESCRIPTION OF OBJECT HEAD

d. **Point.** *Point* endows an object with characteristics to support orthodromic distance calculations. Objects in DAMOC that have locations (bases and installations) are, or are descendants of, this type. It accepts latitude and longitude entries (converting them internally to radians for spherical trigonometric calculations) as well as location-naming information.

OBJECT TYPE: point		ANCESTOR TYPE: link
VARIABLES:		
x	real	the internal spherical equivalent of the place's latitude
y	real	the internal spherical equivalent of the place's longitude
name	string	the common place name
code	string	the code used to identify the place (e.g., a GEOLOC)
FUNCTIONS:		
disto	real	calculates great circle arc distance (nm) from this point to a designated place
next	refpoint	returns the next object in the referenced list
PROCEDURES:		
init	the constructor that converts input values into place variables	

Figure C-5. ATTRIBUTE DESCRIPTION OF OBJECT POINT

e. **Threats.** The information read in for each entry in the THREAT file is used to create *threats* objects. These objects comprise the threat set.

OBJECT TYPE: threats		**ANCESTOR TYPE:** link
VARIABLES:		
name	string	familiar name of threat system
thrtype	integer	internally used threat index (bomber = 2, etc.)
base	refpoint	points to initial location base
range	real	system range
readyrate	real	operational readiness rate for planes (see threat in file annex for ssm and sof)
attrition	real	attrition or use percent
amount	real	initial starting quantity
minimum	real	minimum level of onhand stocks
move	integer[]	move times
change	integer[]	rate change dates
fwd	refpoint[]	stores base pointers used for theater moves
rates	real[]	saves rate change table
FUNCTIONS:		
reaches	boolean	test if target is in range
next	refthreats	gets next object in threats list
PROCEDURES:		
init	constructor used to initialize threat objects	
update	checks to see if move or change should be made	

Figure C-6. ATTRIBUTE DESCRIPTION OF OBJECT THREATS

f. **Threatlist.** *Threatlist* is the list in which all *threat* objects are saved. During processing, threat items are always accessed seriatim.

OBJECT TYPE: threatlist		**ANCESTOR TYPE:** head
FUNCTIONS: first	refthreats	returns first threat system in this list
PROCEDURES: build dump	initiates construction of attacker objects (consistency checks are within threats) prints validated attacker list	

Figure C-7. ATTRIBUTE DESCRIPTION OF OBJECT THREATLIST

g. **Damage.** This object--only one is created during execution--is the damage table. Its purpose is to group all the notional installations and their associated damage profiles.

OBJECT TYPE: damage		**ANCESTOR TYPE:** head
FUNCTIONS: first find	refnotnlinst refnotnlinst	returns the first notional installation in the table looks for a particular notional installation (if not present then find <- null)
PROCEDURES: build breakout dump	this method opens the damage file and reads in the profiles. It also creates the category code (facility identification) list based on the facilities found this method sums all the attacks (sorties, raids, missiles) against installation classes (i.e., notional installations) and prints it by period produces the summary of damage profiles, including the check of facility quantities	

Figure C-8. ATTRIBUTE DESCRIPTION OF OBJECT DAMAGE

h. **Places.** This object type is for the list of attacker bases to which threat systems can be deployed.

OBJECT TYPE: places		ANCESTOR TYPE: head
FUNCTIONS: find first	 refpoint refpoint	 searches for point with requested code returns first member of list
PROCEDURES: build dump	 initiates construction of base list prints places contents	

Figure C-9. ATTRIBUTE DESCRIPTION OF OBJECT PLACES

i. **Notnlinst.** This type encapsulates the information associated with a notional installation of type "instype." The damage profiles (*profile*) against this "instype" are accessed through the *threats* array. Collectively, the *notnlinst* objects comprise the damage table.

OBJECT TYPE: notnlinst		ANCESTOR TYPE: link
VARIABLES: instype sortyalloc threats	 string sortyp refpro-file[]	 the user designated identification code for notional type (e.g., COB for collocated operating bases) an array type used to record how many sorties were launched against this notional instal a pointer array to damage profiles (every threat type need not be present)
FUNCTIONS: next	 refnotnl-inst	 returns the next notional installation in the damage table
PROCEDURES: init2 addsorties attacks	 constructor creates the notional installations information from target cumataks are added to notional installations at the end of an execution the total sorties are rolled up and summarized by period (but only if installation reports are requested)	

Figure C-10. ATTRIBUTE DESCRIPTION OF OBJECT NOTNLINST

j. **Catcodelist.** This object type contains the facility master list. It is derived from the facilities encountered in the damage profiles. It conserves memory and enforces uniformity by using a common label for each facility type.

OBJECT TYPE: catcodelist		ANCESTOR TYPE: head
FUNCTIONS:		
first	refcatcode	this returns the first catcode in the list
add	refcatcode	this routine checks if the catcode is already in the list--if it is not, a new catcode is created and filed in its sorted location
PROCEDURES:		
dump	prints out the contents of the list	

Figure C-11. ATTRIBUTE DESCRIPTION OF OBJECT CATCODELIST

k. **Catcode.** Catcode stands for the category code designation for facilities. While there is no stricture against defining unique facility codes, it is recommended that DAMOC use the JCS 4-character version (for example, the Army uses a 5-character code in its facility component system).[3] *Catcode* objects contain a descriptive title of the catcode, and their relative location in the *catcodelist* set is used as an index in the facility rollup reports.

OBJECT TYPE: catcode		ANCESTOR TYPE: link
VARIABLES:		
code	string	a 4 character code used internally to identify facilities (DAMOC does not validate codes, the user must ensure consistency)
name	string	the long name for a facility
repindx	integer	the relative location of the facility in the reference list
FUNCTIONS:		
next	refcatcode	returns the next catcode in the list
PROCEDURES:		
init	the constructor which initializes the object	

Figure C-12. ATTRIBUTE DESCRIPTION OF OBJECT CATCODE

[3] *Department of Defense Facility Classes and Construction Categories, DOD Instruction 4165.3* (24 October 1978).

l. **Profile.** This object embodies the individual damage profiles. It is accessed through the *threats* array in *notnlinst* objects. It contains the list of facility amounts and damage (*facildam* objects) as well as the attack package sizes that produce the damage.

OBJECT TYPE: profile		ANCESTOR TYPE: head
VARIABLES:		
patks	integer	the number of primary attacks necessary to produce the associated damage
satks	integer	the suppression attack package sizes (planes against air bases and ports)
FUNCTIONS:		
first	refacildam	returns the first facildam object in the template (i.e., damage profiles)
PROCEDURES:		
init2	the constructor method that initializes the object and decodes the primary and secondary attack amounts from input	

Figure C-13. ATTRIBUTE DESCRIPTION OF OBJECT PROFILE

m. **Installation.** The focus of DAMOC is on installations. To whom do they belong? Where are they? What attacks are made against them? What facilities are damaged? To satisfy these demands, this object class incorporates more variable and method attributes than any other class.

OBJECT TYPE: installation		ANCESTOR TYPE: point
VARIABLES:		
ref	string	notional installation ID
nation	char	single character nation/organization indicator (user defined)
passes	real[]	primary attacks made that day, by threat type
suprsatks	real[]	suppression attacks
curatklog	^cumatks	pointer to current phase attack log
initatklog	^cumatks	first log in stack
suprsday	integer	last day either saturated or suppressed (air bases & ports only)
saturated	boolean[]	flags threat type attainment of threshold attack level
airbase	boolean	true if COB, MOB, or other air base
port	boolean	true if SMPORT or LGPORT
unsat	boolean	if "unsaturated" type installation then true
daminfo	refnotnl-inst	points to appropriate notional entry in damage table
FUNCTIONS:		
next	refinstal-lation	returns the next installation in the "front" list
PROCEDURES:		
init		constructor which initializes the many installation attributes
attack		this method determines how many attackers are necessary to saturate or suppress; how many are available, and how they will be allocated
fine		executed when an installation is finished, i.e., saturated
property		returns a list of onhand facility quantities (found in the notional list)
bda		performs the damage assessment
pdreport		checks to see if any attacks have been made against this installation during the period--if yes, then the amounts are added to the cumulative period table
update		if installation is "reconstituted," this method closes and creates attack logs
pmd		performs the postmortem dump (PMD) that prints each attack log phase and damage, hits, and crater summaries

Figure C-14. ATTRIBUTE DESCRIPTION OF OBJECT INSTALLATION

n. **Facildam.** This object type contains the damage information. Each attack *profile* object has a variable number of these objects corresponding to the particular types of facilities that are damaged during an attack.

OBJECT TYPE: facildam		ANCESTOR TYPE: link
VARIABLES:		
catcode	string	the JCS category code (4 characters)
xcatcode	refcatcode	pointer to catcode list with long title
onhand	longint	the installations onhand quantity
damprcnt	real	the percent damaged in full attack
hits	real	the associated number of hits
criticals	real	the number of critical craters to restore a minimum operational strip (MOS)
FUNCTIONS:		
next	refacildam	returns the next facility damage entry in the profile
pavement	boolean	true if the catcode is either 111A, 112A, or 113A
pier	boolean	true if the catcode is 151C
PROCEDURES:		
init2	the constructor which initializes the object and decodes the input record values	

Figure C-15. ATTRIBUTE DESCRIPTION OF OBJECT FACILDAM

o. **Cumatks.** This is a special object type used to record the attacks made against an installation. Except for the possibility of a target being *reconstituted* (i.e., the damage fully repaired), this information could have been part of an *installation* object. Anticipating the possible addition of repair capability being added to the model (directly or indirectly), *cumatks* now enables the model to track successive phases of attacks. "Phase" here means the time from when a target could be attacked to when it is considered fully repaired, thereby initiating a new phase.

OBJECT TYPE: cumatks		**ANCESTOR TYPE:** none
VARIABLES:		
nextlog	refcumatks	points to the next cumatks object found at a target
cumpasses	sortyp	stores the primary attack sorties for this attack phase
cumsuprs	sortys	stores suppression attacks during phase
startday	integer	the day this phase begins
endday	integer	the day this phase ends
FUNCTIONS:		
contrib	boolean	checks start and end days to see if period and phase intersect
PROCEDURES:		
init	the constructor which sets up timing and reference values	
accum	adds current period's running totals to the '0th' column of the pass and suppress arrays	
close	when an installation is reconstituted, a new cumatks is created; this method terminates the prior object	
prime	zeroes the accumulator (running total) portions of the pass and suppress arrays	
range	since start and end dates for the object may not be on period boundaries, this routine determines the period range for the "breakout" reports	
dump	this report is part of the installation pind summary	

Figure C-16. ATTRIBUTE DESCRIPTION OF OBJECT CUMATKS

p. **Area.** This is the set object where *installation* objects are assigned. Note that as the model is presently configured, the order of the installations determines their relative priority as a target.

OBJECT TYPE: area		ANCESTOR TYPE: head
FUNCTIONS: first	refinstal- lation	returns the first installation in the list
PROCEDURES: build		this routine opens the target file and if the data meets certain internal consistency checks, creates installation objects for each valid entry
postmortem		at the end of execution this routine will initiate reporting of damage assessments at installations in the selected set (the default is to report; use '!' in the country string if reports are not wanted)
dump		prints the list of valid installation targets in list order

Figure C-17. ATTRIBUTE DESCRIPTION OF OBJECT AREA

5. **MAIN PROGRAM.** The third software layer of DAMOC is the main program. This code defines the scenario parameters, requests input file identities, initiates appropriate table (damage, catcode) and list build routines, coordinates threat allocation against targets, and defines or invokes the various reports.

6. **PROGRAM LISTINGS.** The actual program listings are found in **Appendices C-1, C-2, and C-3** that accompany this annex.

Blank Page

LAST PAGE OF ANNEX C

APPENDIX C-1

SIMSETx PROGRAM LISTING

Blank Page

SIMSETx PROGRAM LISTING

```
unit SIMSETx;

INTERFACE

TYPE

    reflinkage =    ^linkage;
    reflink    =    ^link;
    refhead    =    ^head;

    linkage      =  object
       suc,pred  :   reflinkage;
       constructor init;
       procedure out;
       end;

    link = object(linkage)
       procedure follow(x:reflinkage);
       procedure precede(x:reflinkage);
       procedure into(s:refhead);
       function next:reflink;
       end;

    head = object(linkage)
       function first: reflinkage;
       function last: reflinkage;
       procedure clear;
       function empty: boolean;
       function cardinal: integer;
       constructor init;
       end;

IMPLEMENTATION

{.................................. ..................LINKAGE...}

    constructor linkage.init;
       begin pred := nil; suc := nil; end;

    procedure linkage.out;
       begin
       pred^.suc := suc; suc^.pred := pred;
       suc := nil; pred := nil;
       end;

{.................................................LINK......}

    procedure link.follow(x:reflinkage);
       begin
       out;
       if x <> nil then
          begin
          if x^.suc <> nil then
              begin
              pred := x;suc := x^.suc;
              suc^.pred := @self;x^.suc := @self;
              end;
          end;
       end;

    procedure link.precede(x:reflinkage);
       begin
       out;
       if x <> nil then
          begin
          if x^.suc <> nil then
             begin
```

```
                suc := x; pred := x^.pred;
                pred^.suc := @self; x^.pred := @self;
                end;
             end;
          end;

      procedure link.into(s:refhead);
          begin precede(s);  end;

      function link.next:reflink;
          begin
          if typeof(suc^) <> typeof(self) then
             next := nil
             else
             next := @suc^;
          end;

   {.............................................HEAD......}

      function head.cardinal: integer;
          var
          i  :  integer;
          p  :  reflinkage;
          begin
          i := 0;
          p := first;
          while p <> nil do
             begin
             i := i + 1;
             p := p^.suc ; if p = @self then p := nil;
             end;
          cardinal := i;
          end;

      function head.first:reflinkage;
          begin
          if not empty then first := suc else first := nil;
          end;

      function head.last:reflinkage;
          begin
          if not empty then last := pred else last := nil;
          end;

      function head.empty:boolean;
          begin   empty := suc = @self;   end;

      procedure head.clear;
          var
          x  :  reflinkage;
          begin
          while first <> nil do
             begin
             x := first;
             x^.out;
             end;
          end;

      constructor head.init;
          begin  suc:=@self;  pred:=@self;  end;

   end {  SIMSET UNIT }.
```

LAST PAGE OF APPENDIX C-1

APPENDIX C-2

COMMZ PROGRAM LISTING

Blank Page

COMMZ PROGRAM LISTING

```
unit commz;

interface

uses simsetx;

const
    nthrt        :    integer = 4;
    maxperiod    :    integer = 10;
    threat : array[1..4] of string[10]=('FIGHTER   ','BOMBER    ',
                                        'SOF       ','SSM       ');
type
    refdamage        =   ^damage;
    refnotnlinst     =   ^notnlinst;
    refprofile       =   ^profile;
    refcatcodelist   =   ^catcodelist;
    refthreatlist    =   ^threatlist;
    refacildam       =   ^facildam;
    refinstallation  =   ^installation;
    refcatcode       =   ^catcode;
    refarea          =   ^area;
    refthreats       =   ^threats;
    refpoint         =   ^point;
    refplaces        =   ^places;
    refcumatks       =   ^cumatks;

    sortyp           =   array[1..4,0..10] of real;
    sortys           =   array[1..2,0..10] of real;
    damrep           =   array[1..75,1..3] of real;
    inventory        =   array[1..75] of real;

    point           =   object(link)
       x,y          :   real;
       name,code    :   string[20];
       constructor init(nm,short,lat,lon:string);
       function disto(dest:refpoint):real;
       function next: refpoint;
       end { point };

    threats        =   object(link)
       name         :   string[12];
       thrtype      :   integer;
       base         :   refpoint;
       range        :   real;
       readyrate    :   real;
       attrition    :   real;
       amount       :   real;
       minimum      :   real;
       move,change:     array[1..3] of integer;
       fwd          :   array[1..3] of refpoint;
       rates        :   array[1..3,1..2] of real;

       constructor init(s,r:string;bases:refplaces;var good:boolean);
       procedure update(dy:integer;var logfyl:text);
       function reaches(tgt:refinstallation):boolean;
       function next: refthreats;
       end { threats };

    places     = object(head)
       procedure build;
       procedure dump;
       function find(itype:string):refpoint;
       function first: refpoint;
       end { places };

    threatlist  = object(head)
       procedure build(bases:refplaces);
```

```
        procedure dump;
        function first: refthreats;
        end { threatlist };

     notnlinst     =   object(link)
        instype    :   string[10];
        sortyalloc :   sortyp;
        threats    :   array[1..4] of refprofile;
        constructor init2(t:string);
        function next: refnotnlinst;
        procedure addsorties(prbeg,prend:integer;var m1:sortyp;var m2:sortys);
        procedure attacks(per:integer);
        end { notnlist };

     damage        =   object(head)
        function first: refnotnlinst;
        procedure build(faclist:refcatcodelist);
        function find(itype:string):refnotnlinst;
        procedure dump(maxcats:integer);
        procedure breakout(nprd:integer)
        end { damage };

     profile       =   object(head)
        patks      :   integer;
        satks      :   integer;
        function first: refacildam;
        constructor init2(s:string);
        end { profile };

     facildam      =   object(link)
        catcode    :   string[4];
        xcatcode   :   refcatcode;
        onhand     :   longint;
        damprcnt   :   real;
        hits       :   real;
        criticals  :   real;
        function next: refacildam;
        function pavement : boolean;
        function pier : boolean;
        constructor init2(st:string);
        end { facildam };


     cumatks       =   object
        nextlog    :   refcumatks;
        cumpasses  :   sortyp;
        cumsuprs   :   sortys;
        startday   :   integer;
        endday     :   integer;
        constructor init(day:integer);
        procedure accum(period:integer);
        procedure close(nxtlog:refcumatks;day:integer);
        procedure prime;
        function contrib(period:integer): boolean;
        procedure range(nper,perlen:integer;daminfo:refnotnlinst;
                        var perstrt,perend:integer);
        procedure dump(supatks:boolean;pr:integer);
        end {cumatks};


     installation  =   object(point)
        ref        :   string[10];
        nation     :   char;
        passes     :   array [1..4] of real;
        suprsatks  :   array [1..2] of real;
        curatklog  :   ^cumatks;
        initatklog :   ^cumatks;
```

```
            suprsday    :   integer;
            saturated   :   array [1..4] of boolean;
            airbase,port:   boolean;
            unsat       :   boolean;
            daminfo     :   refnotnlinst;
            constructor init(var s:refdamage;rec:string;var accept:boolean);
            function next: refinstallation;
            procedure attack(var amt:real;var log:text;thrtyp,dy,npr,
                         sprcycle:integer;tn:string);
            procedure fine(thrt,dy:integer);
            procedure property(var assets:inventory);
            procedure bda(ip,mf:integer;atks:refcumatks;
                          var assets:inventory;var results:damrep);
            procedure pdreport(period,day,maxf:integer;var results:damrep);
            procedure update(pr,day:integer;reset:boolean);
            procedure pmd(fx,pr,periodlen:integer;clist:refcatcodelist);
            end { installation };

         area      =   object(head)
            function first: refinstallation;
            procedure build(d:refdamage);
            procedure postmortem(nfacs,nprd,lenprd:integer;
                                 ftab:refcatcodelist;mask:string);
            procedure dump;
            end { area };

         catcode    =   object(link)
            code    :   string[4];
            name    :   string[20];
            repindx :  integer;
            constructor init(c,n:string);
            function next: refcatcode;
            end { catcode };

         catcodelist    =   object(head)
            function first: refcatcode;
            function add(s:string) : refcatcode;
            procedure dump;
            end { catcodelist };

      implementation

      {..................POINT  METHODS....................................}

         constructor point.init(nm,short,lat,lon:string);
            var
            xd,xm,xs,yd,ym,ys,degperad : real;
            c  :  integer;
            begin
            link.init;
            val(copy(lat,1,2),xd,c);
            val(copy(lat,3,2),xm,c);
            val(copy(lon,1,3),yd,c);
            val(copy(lon,4,2),ym,c);
            name := nm; code := short;
            degperad := 360/(2 * Pi);
            x := (xd+xm/60) / degperad;
            y := (yd+ym/60) / degperad;
            if copy(lat,7,1) = 'S' then x := - x;
            if copy(lon,8,1) = 'W' then y := - y;
            end;

            function point.disto(dest:refpoint):real;
               var
               arc,delta,cose,radist : real;
               begin
               delta := Abs(y - dest^.y);
```

```
        if delta > Pi then delta := (2*Pi) - delta;
        cose := sin(x)*sin(dest^.x) + cos(x)*cos(dest^.x)*cos(delta);
        radist := (Pi/2) - arctan( cose / sqrt( 1 - cose*cose) );
        arc := radist * (360/(2 * Pi) ) * 60; {minutes of arc}
     {  disto := 1.852 * arc;  kilometer conversion of 1' arc}
        disto := arc;          {distance in nautical miles}
        end;

      function point.next: refpoint;
         var lnk  :  reflink;
         begin  lnk := link.next;  next := @lnk^;  end;


{..................PLACES  METHODS...................................}

      procedure places.build;
         var
         fn : text; fyle : string[20]; buf : string[80];
         p  :  refpoint;
         begin
         write(' enter filename of threat bases--> ');
         readln(fyle); assign(fn, fyle); reset(fn);
         while not eof(fn) do
            begin
            readln(fn,buf);
            new(p);
            if p^.init(copy(buf,1,20),copy(buf,25,5),
                        copy(buf,41,7),copy(buf,51,8)) then
               p^.into(@self)
               else
               writeln('..base rejected --',buf);
            end;
         close(fn);
         end;

      function places.first: refpoint;
         var lkg  :  reflinkage;
         begin  lkg := head.first;  first := @lkg^;  end;

   function places.find(itype:string):refpoint;
      var
      p  :  refpoint;
      begin
      find := nil;
      p := first;
      while p <> nil do
            if p^.code = itype then
               begin
               find := p;
               p := nil;
               end
               else
               p := p^.next;
      end;


      procedure places.dump;
        var t : refpoint; cnt : integer;
        begin
        writeln('.......attack bases defined');
        t := first;
        while t <> nil do
          begin
          writeln(t^.name:25,' [',t^.code,']  ',
                t^.x:10:5,t^.y:10:5);
          t := t^.next;
          end;
```

```
         writeln;
         end;



{..................THREATS METHODS...................................}


   constructor threats.init(s,r:string;bases:refplaces;var good:boolean);
      var typ : string[10]; i,err : integer;
      begin
      link.init;
      base := bases^.find(copy(s,31,5));
      if base = nil then
         begin
         writeln('...threat start base error - ',s);
         good := false;
         end
         else
         begin
         typ := copy(s,21,10); thrtype := -1;
         for i := 1 to nthrt do if typ = threat[i] then thrtype := i;
         if thrtype > 0 then
            begin
            good := true;
            name := copy(s,1,20);
            val(copy(s,36,5),range,err);
            val(copy(s,41,5),amount,err);
            val(copy(s,46,5),minimum,err);
            readyrate := 0.0;   attrition := 0.0;
            for i := 1 to 3 do
               begin
               val(copy(s,51+10*(i-1),5),move[i],err);
               if move[i] > 0 then
                  begin
                  fwd[i] := bases^.find(copy(s,56+10*(i-1),5));
                  if fwd[i] = nil then
                     begin
                     writeln('  - redeploy error in ',s);
                     good:=false;
                     end;
                  end;
               end;
            for i := 1 to 3 do
                begin
                val(copy(r,11+15*(i-1),5),change[i],err);
                if change[i] > 0 then
                   begin
                   val(copy(r,16+15*(i-1),5),rates[i,1],err);
                   val(copy(r,21+15*(i-1),5),rates[i,2],err);
                   if rates[i,1]*rates[i,2] > 1 then
                      begin
                      writeln('   - rate range error in ',name);
                      good := false;
                      end;
                   end;
                end;
            end
            else
            good  := false;
         end;
      end;

    function threats.reaches(tgt:refinstallation):boolean;
      begin
      if base^.disto(tgt) < range then reaches := true
         else reaches := false;
```

```
            end;

         procedure threats.update(dy:integer;var logfyl:text);
            var i : integer;
            begin
            for i := 1 to 3 do
               begin
               if dy = move[i] then
                  begin
                  writeln(logfyl,'---[',dy:2,']---',name,
                     ' moves from ',base^.name,' to ',fwd[i]^.name);
                  base := fwd[i];
                  end;
               if dy = change[i] then
                  begin
                  readyrate := rates[i,1];
                  attrition := rates[i,2];
                  end;
               end;
            end;

         function threats.next: refthreats;
            var lnk  :  reflink;
            begin  lnk := link.next;  next := @lnk^;  end;



{..................THREATLIST METHODS.................................}


         procedure threatlist.build(bases:refplaces);
            var
            fn : text; fyle : string[20]; buf1,buf2 : string[8C];
            t  :  refthreats;   ok   :  boolean;
            begin
            write(' enter filename of threat systems--> ');
            readln(fyle); assign(fn, fyle); reset(fn);
            while not eof(fn) do
               begin
               readln(fn,buf1);readln(fn,buf2);
               new(t,init(buf1,buf2,bases,ok));
               if ok then t^.into(@self)
                  else
                  writeln('..threat rejected --',buf1);
               end;
            close(fn);
            end;

         function threatlist.first: refthreats;
            var lkg  :  reflinkage;
            begin  lkg := head.first;  first := @lkg^;  end;


         procedure threatlist.dump;
           var t : refthreats; cnt : integer;
           begin
           writeln('.......threat definition');
           t := first;
           while t <> nil do
             begin
             writeln(t^.name:25,' [',threat[t^.thrtype],']  ',
                   t^.range:6:1);
             t := t^.next;
             end;
           writeln;
           end;
```

```
    {..................DAMAGE METHODS....................................}

        function damage.find(itype:string):refnotnlinst;
           var
           ri  :  refnotnlinst;
           begin
           find := nil;
           ri := first;
           while ri <> nil do
                 if ri^.instype = itype then
                    begin
                    find := ri;
                    ri := nil;
                    end
                    else
                    ri := ri^.next;
           end;

        procedure damage.build(faclist:refcatcodelist);
           var
              fn  :  text;
              i,it:  integer;
              fyle  :  string[20];
              iobuf  :  string[80];
              facrec :  refacildam;
              thrt   :  refprofile;
              tinst  :  refnotnlinst;
              thrtyp :  string[10];
           begin
           write(' enter filename of installation-attack profiles--> ');
           readln(fyle); assign(fn, fyle); reset(fn);
           while not eof(fn) do
              begin
              readln(fn,iobuf);
              if iobuf[1] = '*' then
                 begin
                 tinst := find(copy( iobuf,2,10 ));
                 if tinst = nil then
                    begin
                    writeln('instalprofile created for [',copy(iobuf,2,10),']');
                    tinst:=new(refnotnlinst,init2( iobuf ));
                    tinst^.into(@self);
                    end;
                 thrtyp := copy(iobuf,26,10);
                 it:=0;for i:= 1 to nthrt do if thrtyp = threat[i] then it := i;
                 if it = 0 then
                    writeln('----invalid threat type -- ',thrtyp)
                    else
                    if tinst^.threats[it] = nil then
                       begin
                       tinst^.threats[it]:=
                           new(refprofile,init2(iobuf));
                       thrt:=tinst^.threats[it];
                       end
                       else
                       begin
                       thrt := nil;
                       writeln('...duplicate threat encountered-',iobuf);
                       end;
                 end
                 else
                 if thrt <> nil then
                    begin
                    facrec:=new(refacildam,init2(iobuf));
                    facrec^.into(thrt);
                    facrec^.xcatcode := faclist^.add(iobuf);
                    end;
```

```
        end;
      close(fn);
      end;

    function damage.first: refnotnlinst;
        var lkg  :  reflinkage;
        begin  lkg := head.first;  first := @lkg^;  end;


    procedure damage.breakout(nprd:integer);
       var
       instclass  :  refnotnlinst;
       begin
       writeln;
       writeln('Sortie/Installation Breakout: .........');
       writeln;
       instclass := first;
       while instclass <> nil do
          begin
          instclass^.attacks(nprd);
          instclass := instclass^.next;
          end;
       end;


       procedure damage.dump(maxcats:integer);
          var
             x : refnotnlinst;
             f : refacildam;
             assetchk  :  array [1..100] of longint;
             i,j : integer;
          begin
          writeln('#==DAMAGE TEMPLATE INPUT SUMMARY!');
          x := first;
          while x <> nil do
             begin
             writeln(x^.instype:15);
             for i := 1 to nthrt do
                if x^.threats[i] <> nil then
                   writeln('       ',threat[i]:12,'  [',
                    x^.threats[i]^.patks:5,x^.threats[i]^.satks:5,']',
                    x^.threats[i]^.cardinal:10,'  facilities damaged.');
             for i := 1 to maxcats do assetchk[i] := 0;
             for i := 1 to nthrt do
                 begin
                 f := x^.threats[i]^.first;
                 while f <> nil do
                     begin
                     j := f^.xcatcode^.repindx;
                     if assetchk[j] = 0 then
                          assetchk[j]:=f^.onhand
                          else
                          if assetchk[j] <> f^.onhand then
                               writeln('    * onhand inconsistency for ',
                                  f^.catcode,'--[',assetchk[j]:10,
                                  ' <> ',f^.onhand:10,']');
                     f := f^.next;
                     end;
                 end;
             x := x^.next;
             end;
          end;

{..................NOTNLINST METHODS....................................}

    constructor notnlinst.init2(t:string);
       var i,j,err : integer;
```

```
              begin
              link.init;
              instype := copy(t,2,10);
              for i := 1 to 4 do
                 begin
                 threats[i] := nil;
                 for j := 1 to maxperiod do
                    sortyalloc[i,j] := 0.0;
                 end;
              end;

           function notnlinst.next: refnotnlinst;
               var lnk  :  reflink;
               begin  lnk := link.next;  next := @lnk^;  end;



           procedure notnlinst.addsorties(prbeg,prend:integer;var m1:sortyp;var
    m2:sortys);
              var i,j  :  integer;
              begin
              for j := prbeg to prend do
                 for i := 1 to 4 do
                    begin
                    sortyalloc[i,j] := sortyalloc[i,j] + m1[i,j];
                    if i < 3 then
                       sortyalloc[i,j] := sortyalloc[i,j] + m2[i,j];
                    end;
              end;

            procedure notnlinst.attacks(per:integer);
               var  i,j : integer;s : real;
               begin
               writeln;write('<< ',instype,' >>');
              {writeln;
               for i := 1 to nthrt do
                   begin
                   write(threat[i]);
                   for j := 1 to per do write(sortyalloc[i,j]:8:1);
                   writeln;
                   end;}
               for j:= 1 to per do
                   begin
                   s:= sortyalloc[1,j] + sortyalloc[2,j] + sortyalloc[3,j]
                         +sortyalloc[4,j];
                   write(s:8:1);
                   end;
               writeln;
               end;

    {..................FACILDAM METHODS.....................................}


         constructor facildam.init2(st:string);
            var err : integer;
            begin
            link.init;
            catcode := copy(st,26,4);
            val(copy(st,31,10),onhand,err);
            val(copy(st,41,10),damprcnt,err); damprcnt:= 0.01 * damprcnt;
            val(copy(st,51,10),hits,err);
            val(copy(st,61,10),criticals,err);
            end;

           function facildam.next: refacildam;
               var lnk  :  reflink;
               begin  lnk := link.next;  next := @lnk^;  end;
```

```
      function facildam.pavement: boolean;
        begin
        if (catcode='111A') or (catcode='112A') or (catcode='113A') then
           pavement := true
           else
           pavement := false;
        end;

      function facildam.pier: boolean;
        begin
        if (catcode='151C')  then
           pier := true
           else
           pier := false;
        end;



  {................... CUMATAKS METHODS .......................................}


       constructor cumatks.init(day:integer);
          var i,j  :  integer;
          begin
          nextlog := nil;
          startday := day;
          endday := 999;
          for i := 0 to 10 do
             begin
             for j := 1 to 4 do cumpasses[j,i] := 0.0;
             cumsuprs[1,i] := 0.0;
             cumsuprs[2,i] := 0.0;
             end;
          end;

       procedure cumatks.accum(period:integer);
          var i : integer;
          begin
          for i := 1 to 4 do
             cumpasses[i,0] := cumpasses[i,0] + cumpasses[i,period];
          cumsuprs[1,0] := cumsuprs[1,0] + cumsuprs[1,period];
          cumsuprs[2,0] := cumsuprs[2,0] + cumsuprs[2,period];
          end;

       function cumatks.contrib(period:integer): boolean;
          var i : integer; test  :  real;
          begin
          test := cumsuprs[1,period] + cumsuprs[2,period];
          for i := 1 to nthrt do
              test := test + cumpasses[i,period];
          if test > 0.0 then
             contrib := true
             else
             contrib := false;
          end;

       procedure cumatks.close(nxtlog:refcumatks;day:integer);
          begin
          nextlog := nxtlog;
          endday := day;
          end;


       procedure cumatks.prime;
          var j : integer;
          begin
```

```
          for j := 1 to 4 do cumpasses[j,0] := 0.0;
          cumsuprs[1,0] := 0.0;
          cumsuprs[2,0] := 0.0;
          end;

       procedure cumatks.range(nper,perlen:integer;daminfo:refnotnlinst;
                               var perstrt,perend:integer);
          begin
          perstrt :=    ( (startday-1) div perlen ) +1;
          if endday = 999 then
             perend := nper
             else
             perend :=  ((endday-1) div perlen) + 1;
          daminfo^.addsorties(perstrt,perend,cumpasses,cumsuprs);
          end;


       procedure cumatks.dump(supatks:boolean;pr:integer);
         var i,j  :  integer;
         begin
         for j := 1 to 4 do
            begin
            write('[',threat[j],'/p] ');
            for i := 1 to pr do write(cumpasses[j,i]:5:1);writeln;
            end;
         if supatks then
            for j := 1 to 2 do
               begin
               write('[',threat[j],'/s] ');
               for i := 1 to pr do write(cumsuprs[j,i]:5:1);writeln;
               end;
         writeln;
         end;


{..................INSTALLATION METHODS.....................................}

    constructor installation.init(var s:refdamage;
                                  rec:string;var accept:boolean);
       var
       instype :  string;
       indam   :  refnotnlinst;
       c,i,p   :  integer;
       begin
       instype := copy(rec,30,10);
       indam :=  s^.find(instype);
       if indam <> nil then
          begin
          point.init(copy(rec,1,20),'',copy(rec,50,7),copy(rec,60,8));
          ref   := instype;   daminfo  := indam;     nation := rec[25];
          accept := true;      suprsday := 0;
          for i := 1 to nthrt do
             begin passes[i] := 0.0; saturated[i] := false; end;
          if rec[27] = '*' then unsat := true else unsat := false;
          suprsatks[1] := 0.0; suprsatks[2] := 0.0;
          initatklog := new(refcumatks,init(1));
          curatklog := initatklog;
          if indam^.threats[1]^.satks > 0 then
             begin
             if (ref = 'NCAF      ') or
                (ref = 'COB       ') or
                (ref = 'MOB       ') then
                begin airbase := true; port := false; end
                else
                if (ref = 'LGPORT    ') or
                   (ref = 'SMPORT    ') then
                   begin port := true; airbase := false; end
```

```
                    else
                    begin
                    airbase := false;
                    port := false;
                    end;
                end
                else
                begin
                airbase := false;
                port := false;
                end;
            end
            else
            accept := false;
          end;

        function installation.next: refinstallation;
            var lnk  :  reflink;
            begin  lnk := link.next;  next := @lnk^;  end;


        procedure installation.attack(var amt:real;var log:text;thrtyp,dy,npr,
                        sprcycle:integer;tn:string);
           var
           pcntair:  real;
           needs  :  real;
           begin
           if daminfo^.threats[thrtyp] <> nil then
             begin

               if (not saturated[thrtyp]) then
                  begin
                  if unsat then
                     needs := daminfo^.threats[thrtyp]^.patks
                     else
                     begin
                     if thrtyp > 2 then
                        needs := daminfo^.threats[thrtyp]^.patks
                                    - passes[thrtyp]
                        else
                        begin
                        pcntair :=
                          ( passes[1] / daminfo^.threats[1]^.patks )   +
                          ( passes[2] / daminfo^.threats[2]^.patks );
                        needs := (1-pcntair)*daminfo^.threats[thrtyp]^.patks;
                        end;
                     end;
                  writeln(log,dy:2,' /',name,needs:4:1,
                                  '-',tn,amt:5:1);
                  if needs < amt then
                     begin
                     amt := amt - needs;
                     passes[thrtyp] := 0;
                     if thrtyp < 3 then passes[3 - thrtyp] := 0;
                     curatklog^.cumpasses[thrtyp,npr] :=
                          curatklog^.cumpasses[thrtyp,npr] + needs;
                     fine(thrtyp,dy);
                     end
                     else
                     begin
                     passes[thrtyp]:= passes[thrtyp] + amt;
                     curatklog^.cumpasses[thrtyp,npr] :=
                          curatklog^.cumpasses[thrtyp,npr] + amt;
                     amt := 0;
                     end;
                  end
                  else
```

```
                  if (airbase or port) and (thrtyp <= 2) then
                     begin
                     if (dy - suprsday) >= sprcycle then
                        begin
                        pcntair :=
                          ( (suprsatks[1])/daminfo^.threats[1]^.satks )   +
                          ( (suprsatks[2])/daminfo^.threats[2]^.satks ) ;
                        needs := (1-pcntair)*daminfo^.threats[thrtyp]^.satks;
                        writeln(log,dy:2,' /',name,needs:4:1,
                                    ' (sup) ',tn,amt:5:1);
                        if (0 < needs) then
                           begin
                           if (needs < amt) then
                              begin
                              amt := amt - needs;
                              suprsday := dy;
                              suprsatks[1] := 0.0;
                              suprsatks[2] := 0.0;
                              curatklog^.cumsuprs[thrtyp,npr] :=
                                 curatklog^.cumsuprs[thrtyp,npr] + needs;
                              end
                              else
                              begin
                              suprsatks[thrtyp] := suprsatks[thrtyp] + amt;
                              curatklog^.cumsuprs[thrtyp,npr] :=
                                      curatklog^.cumsuprs[thrtyp,npr] + amt;
                              amt := 0;
                              end;
                           end;
                        end;
                     end;
               end;
            end;

         procedure installation.fine(thrt,dy:integer);
            begin
            if not great then saturated[thrt] := true;
            if thrt < 3 then
               begin
               suprsday := dy;
               suprsatks[1] := 0;
               suprsatks[2] := 0;
               saturated[3-thrt] := saturated[thrt];
               end;
            end;

         procedure installation.update(pr,day:integer;reset:boolean);
            var t : integer;
            newatklog  :  refcumatks;
            begin
            if reset then
               begin
               newatklog := new(refcumatks,init(day));
               curatklog^.close( newatklog,day);
               curatklog := newatklog;
               for t := 1 to nthrt do saturated[t] := false;
               end;
            end;



         procedure installation.property(var assets:inventory);
            var
            tgtfac    :   refacildam;
            findx,nt  :   integer;
            begin
            for nt := 1 to nthrt do
```

```
          begin
          if daminfo^.threats[nt] <> nil then
             tgtfac := daminfo^.threats[nt]^.first
             else
             tgtfac := nil;
          while tgtfac <> nil do
             begin
             findx := tgtfac^.xcatcode^.repindx;
             if assets[findx] = 0.0 then
                assets[findx] := tgtfac^.onhand;
             tgtfac := tgtfac^.next;
             end;
          end;
       end {property};


     procedure installation.pdreport(period,day,maxf:integer;
                                     var results:damrep);
        var atkrec    : refcumatks;
            fassets   : inventory;
            i,j       : integer;
            damage    : damrep;
        begin
        for i := 1 to maxf do
           begin
           fassets[i] := 0.0;
           damage[i,1] := 0.0;
           damage[i,2] := 0.0;
           damage[i,3] := 0.0;
           end;
        property(fassets);
        atkrec := initatklog;
        while atkrec <> nil do
           begin
           if atkrec^.contrib(period) then
              begin
              bda(period,maxf,atkrec,fassets,damage);
              for i := 1 to maxf do
                 if fassets[i] > 0.0 then
                    for j := 1 to 3 do
                       begin
                       results[i,j]:=results[i,j]+damage[i,j];
                       damage[i,j] := 0;
                       end;
              end;
           atkrec := atkrec^.nextlog;
           end;
        end;


       procedure installation.bda(ip,mf:integer;atks:refcumatks;
                                  var assets:inventory;
                                  var results:damrep);
          var
          factr          :   array [1..4] of real;
          factrs         :   array [1..2] of real;
          tgtfacil       :   refacildam;
          damchk         :   inventory;
          i,nt,findx     :   integer;
          pfactr         :   real;

          begin
          for i := 1 to mf do damchk[i] := 0.0;
          for nt := 1 to nthrt do
             begin
             if (daminfo^.threats[nt] <> nil) then
                begin
```

```
                factr[nt] :=
                    (atks^.cumpasses[nt,ip]) / daminfo^.threats[nt]^.patks;
                pfactr :=
                    (atks^.cumpasses[nt, 0]) / daminfo^.threats[nt]^.patks;
                if (airbase or port) and (nt < 3) then
                  begin
                  if (daminfo^.threats[nt] <> nil) and
                     (daminfo^.threats[nt]^.satks > 0) then
                        factrs[nt] :=
                         (atks^.cumsuprs[nt,ip]) /
                            daminfo^.threats[nt]^.satks
                        else
                        factrs[nt] := 0.0;
                  end;
                tgtfacil := daminfo^.threats[nt]^.first;
                while tgtfacil <> nil do
                  begin
                  findx := tgtfacil^.xcatcode^.repindx;
                  results[findx,1] := results[findx,1]
                    + factr[nt] * tgtfacil^.damprcnt * assets[findx];
                  damchk[findx] := damchk[findx]
                    + pfactr * tgtfacil^.damprcnt * assets[findx];
                  results[findx,2] := results[findx,2]
                    + factr[nt] * tgtfacil^.hits;
                  results[findx,3] := results[findx,3]
                    + factr[nt] * tgtfacil^.criticals;
                  if nt < 3 then
                    begin
                    if factrs[nt] > 0 then
                      begin
                      if airbase then
                        begin
                        if tgtfacil^.pavement then
                          begin
                          results[findx,3] := results[findx,3]
                             + factrs[nt] * tgtfacil^.criticals;
                          results[findx,2] := results[findx,2]
                             + factrs[nt] * tgtfacil^.hits;
                          end;
                        end
                        else
                        if port then
                          begin
                          if tgtfacil^.pier then
                            begin
                            results[findx,2] := results[findx,2]
                                  + factrs[nt] * tgtfacil^.hits;
                            end;
                          end;
                      end;
                    end;
                  tgtfacil := tgtfacil^.next;
                  end;
                end;
            end {nt loop};
         for i := 1 to mf do
            if assets[i] <> 0 then
               begin
               if (damchk[i] + results[i,1] > assets[i] ) then
                  results[i,1] := assets[i] - damchk[i];
               if results[i,1] < 0 then
                  results[i,1] := 0.0;
               end;
         atks^.accum(ip);
         end;
```

```
procedure installation.pmd(fx,pr,periodlen:integer;
                           clist:refcatcodelist);
  var
  totdam        :  array [1..10] of damrep;
  check         :  inventory;
  ip,findx      :  integer;
  stper,endper  :  integer;
  nt,i,j        :  integer;
  catcd         :  refcatcode;
  phase         :  refcumatks;

  begin
  writeln;writeln('++++++Installation summary for ',
           name:20,'++++++++++++++');writeln;
  for i := 1 to pr do
     for j := 1 to fx do
        begin
        totdam[i,j,1] := 0.0;
        totdam[i,j,2] := 0.0;
        totdam[i,j,3] := 0.0;
        end;

  for i := 1 to fx do check[i] := 0.0;
  property(check);
  phase := initatklog;
  while phase <> nil do
    begin
    phase^.dump( (airbase or port), pr );
    phase^.prime;
    phase^.range(pr,periodlen,daminfo,stper,endper);
    for ip := stper to endper do
       begin
       bda(ip,fx,phase,check,totdam[ip]);
       end;
    phase := phase^.nextlog;
    end;
 catcd := clist^.first;
 writeln;writeln('FACILITY DAMAGE');
 write('     Facility     Catcode');
   for i := 1 to pr do write(' period',i:3); writeln;
 write('---------------- -------');
   for i := 1 to pr do write(' ---------'); writeln;
 while catcd <> nil do
    begin
    findx := catcd^.repindx;
    if check[findx] > 0 then
       begin
       write(catcd^.name:20,catcd^.code:4);
       for i := 1 to pr do write(totdam[i,findx,1]:10:1);
       writeln;
       end;
    catcd := catcd^.next;
    end;
 catcd := clist^.first;
 writeln;writeln('FACILITY HITS');
 write('     Facility     Catcode');
   for i := 1 to pr do write(' period',i:3); writeln;
 write('---------------- -------');
   for i := 1 to pr do write(' ---------'); writeln;
 while catcd <> nil do
    begin
    findx := catcd^.repindx;
    if check[findx] > 0 then
       begin
       write(catcd^.name:20,catcd^.code:4);
       for i := 1 to pr do write(totdam[i,findx,2]:10:1);
       writeln;
```

```
                end;
             catcd := catcd^.next;
             end;
          if airbase then
             begin
             catcd := clist^.first;
             writeln;writeln;writeln('CRITICAL CRATERS');
             while catcd <> nil do
                if catcd^.code < '120A' then
                   begin
                   findx := catcd^.repindx;
                   if check[findx] > 0 then
                      begin
                      write(catcd^.name:20,catcd^.code:4);
                      for i := 1 to pr do write(totdam[i,findx,3]:10:1);
                      writeln;
                      end;
                   catcd := catcd^.next;
                   end
                   else
                   catcd := nil;
             end;
          end;

{..................PROFILE METHODS....................................}

       constructor profile.init2(s:string);
          var err : integer;
          begin
          head.init;
          val(copy(s,41,5),patks,err);
          if err<>0 then
             begin
             writeln('primary attack err ',s);
             patks := 0;
             end;
          val(copy(s,46,5),satks,err);
          if err<>0 then
             begin
             writeln('secondary attack err ',s);
             satks := 0;
             end;
          end;

       function profile.first: refacildam;
           var lkg  :  reflinkage;
           begin  lkg := head.first;  first := @lkg^;  end;


{..................AREA METHODS....................................}

       function area.first: refinstallation;
           var lkg  :  reflinkage;
           begin  lkg := head.first;  first := @lkg^;  end;


       procedure area.build(d : refdamage);
            var
               fn     :  text;
               valid  :  boolean;
               fyle   :   string[20];
               iobuf  :  string[80];
               inst   :  refinstallation;
            begin
            write(' enter filename of target installations --> ');
            readln(fyle); assign(fn, fyle); reset(fn);
            while not eof(fn) do
```

```
                begin
                readln(fn,iobuf);
                new(inst,init(d,iobuf,valid));
                if not valid then
                   writeln('..no ref-install for ',iobuf:40)
                   { dispose(inst) }
                   else
                   inst^.into(@self);
                end;
             close(fn);
             end;

      procedure area.postmortem(nfacs,nprd,lenprd:integer;
                               ftab:refcatcodelist;mask:string);
         var
         target :  refinstallation;
         begin
         target := first; write('#');
         while target <> nil do
            begin
            if (pos(target^.nation,mask) > 0 ) or (mask[1] = '*') then
              target^.pmd(nfacs,nprd,lenprd,ftab);
            target := target^.next;
            end;
         end;

       procedure area.dump;
         var c : refinstallation;cnt : integer;
         begin
         writeln('#Regional installations in priority order');
         c := first; cnt := 0;
         while c <> nil do
           begin
           cnt := cnt + 1;
           writeln('(',cnt:5,')  ',c^.name:30,' [Nat=',c^.nation,']  ',
                  c^.ref:15);
           c := c^.next;
           end;
         writeln;
         end;

{..................CATCODE METHODS....................................}

       constructor catcode.init(c,n:string);
         begin
         link.init;
         code := c; name := n;
         end;

       function catcode.next: refcatcode;
          var lnk  :  reflink;
          begin  lnk := link.next;  next := @lnk^;  end;



{..................CATCODELIST METHODS....................................}

       function catcodelist.first: refcatcode;
          var lkg  :  reflinkage;
          begin  lkg := head.first;  first := @lkg^;  end;


        function catcodelist.add(s:string) : refcatcode;
          var
          c1,c2 : refcatcode;
          cd    : string[4];
          begin
```

```
          cd := copy(s,26,4);
          if empty then
             begin
             c2 := new(refcatcode,init(cd,copy(s,3,20)));
             c2^.into(@self);
             add := c2;
             end
             else
             begin
             c1 := first;
             while c1 <> nil do
                begin
                if c1^.code < cd then
                   begin
                   c1 := c1^.next;
                   if c1 = nil then
                      begin
                      c2 := new(refcatcode,init(cd,copy(s,3,20)));
                      c2^.into(@self);
                      add := c2;
                      end;
                   end
                   else
                   if c1^.code > cd then
                      begin
                      c2 := new(refcatcode,init(cd,copy(s,3,20)));
                      c2^.precede(c1);
                      add := c2;
                      c1 := nil;
                      end
                      else
                      begin
                      add := c1;
                      c1 := nil;
                      end;
                end;
             end;
          end;

       procedure catcodelist.dump;
          var c  :  refcatcode; cnt : integer;
          begin
          writeln('#REFERENCE JCS CATCODE LIST:');
          c := first;
          cnt := 0;
          while c <> nil do
             begin
             cnt := cnt + 1; c^.repindx := cnt;
             writeln('(',cnt:5,')  ',c^.code,'----',c^.name);
             c := c^.next;
             end;
          end;

end.
```

Blank Page

LAST PAGE OF APPENDIX C-2

APPENDIX C-3

DAMOC PROGRAM LISTING

Blank Page

DAMOC PROGRAM LISTING

```
program damoc;

uses simsetx,commz,dos;


{==========DAMOC========== MAIN PROGRAM =================(8 MAY 91)==}


var
   log                 :  text;
   front               :  area;
   dtable              :  refdamage;
   instclass           :  refnotnlinst;
   ftable              :  refcatcodelist;
   facil               :  refcatcode;
   btable              :  refplaces;
   ttable              :  refthreatlist;
   thrtgrp             :  refthreats;
   itgt                :  refinstallation;
   day,maxday          :  integer;
   period,nperiod      :  integer;                { maxperiod <= 10 }
   i,j,xt              :  integer;
   maxfac,facindex     :  integer;                  { maxfac <= 75 }
   avail,needs         :  real;
   factor,factorsup    :  real;
   double,sofrq        :  integer;
   ssmfrq,sprsdays     :  integer;
   reconst,reconstno   :  integer;
   sorties             :  array [1..180,1..4] of real;  {maxday <= 180}
   reportbl,totals     :  damrep;
   rollup,rpis         :  inventory;
   countries           :  string[10];
   repinstal           :  boolean;
   minutes             :  real;


   procedure space;
       begin
       writeln('    Avail Memory (heap) = ',memavail);
       end;


   function elapsed(min:real):real;
   var  h,m,s,s1  :  word;min2  :  real;
   begin
   gettime(h,m,s,s1);
   min2 := (60.0*h + 1.0*m + s/60.0);
   if min = 0 then
      writeln(', time is ',h:2,':',m:2,':',s:2)
      else
      writeln('  [elapsed time is ',min2-min:7:3,' minutes]');
   elapsed := min2;
   end;

begin

{-------------------------------------------Scenario Definition------}

write(' heap memory =',memavail:10);
minutes := elapsed(0.0);
write('Enter the number of days in the scenario ->');
     readln(maxday);
write('Select country codes ( a "*" means all included-->');
     readln(countries);
write('Length of period (and report cycle) -->');
     readln(period);
     if (maxday/period) > 10 then
```

```
            begin
            writeln(' .. period maxday overflow.');
            halt;
            end;
    write('Enter days of double sorties & suppression period-->');
         readln(double,sprsdays);
    write('Enter SOF & SSM frequencies ->');
         readln(sofrq, ssmfrq);
    write('Enter reconstitution period and number-->');
         readln(reconst, reconstno);

    {-------------------------------------------Table Construction-----.}

    ftable := new(refcatcodelist,init);
    btable := new(refplaces,init);
    btable^.build;                      btable^.dump;
    ttable := new(refthreatlist,init);
    ttable^.build(btable);              ttable^.dump;
    dtable := new(refdamage,init);
    dtable^.build(ftable);
       ftable^.dump;    maxfac := ftable^.cardinal;
       dtable^.dump(maxfac);
    front.init;
    front.build(dtable);
       front.dump;

    assign(log,'sortie.log');rewrite(log);

    {--------------------------------PARAMETERS & INITIALIZATION......}

    nperiod := 1;
    if (pos('!',countries) > 0 ) then
       repinstal := false
       else
       repinstal := true;

    for i := 1 to maxfac do
       begin
       rollup[i] := 0.0;
       rpis[i] := 0.0;
       for j := 1 to 3 do
          begin
          totals[i,j] := 0.0;
          reportbl[i,j] := 0.0;
          end;
       end;

    itgt := front.first;
       while itgt <> nil do
          begin
          if (pos(itgt^.nation,countries) > 0) or (countries[1] = '*') then
                begin
                itgt^.property(rpis);
                for i := 1 to maxfac do
                   begin
                   rollup[i] := rollup[i] + rpis[i];
                   rpis[i] := 0.0;
                   end;
                end;
          itgt := itgt^.next;
          end;

    for i := 1 to nthrt do for j := 1 to maxday do sorties[j,i] := 0.0;

    {..............start day by day simulation.......................}

    for day := 1 to maxday do
```

```
        begin
        write('=====day(',day,')');minutes := elapsed(minutes);

        {...attack process======================================}

        thrtgrp := ttable^.first;

        while thrtgrp <> nil do
           begin
           xt := thrtgrp^.thrtype;
           thrtgrp^.update(day,log);
           case xt of
      {ftr}
              1: if thrtgrp^.readyrate > 0 then
                 begin
                 avail := thrtgrp^.amount * thrtgrp^.readyrate
                          * (1.0 - thrtgrp^.attrition);
                 thrtgrp^.amount := (1.0 - thrtgrp^.attrition) * thrtgrp^.amount;
                 if day <= double then
                    begin
                    avail := avail + thrtgrp^.amount * thrtgrp^.readyrate
                          * (1.0 - thrtgrp^.attrition);
                    thrtgrp^.amount := (1.0 - thrtgrp^.attrition) * thrtgrp^.amount;
                    end;
                 if thrtgrp^.amount < thrtgrp^.minimum then
                        thrtgrp^.amount := thrtgrp^.minimum;
                 end
                 else
                 avail := 0;

      {bmbr}
              2: if thrtgrp^.readyrate > 0 then
                 begin
                 avail := thrtgrp^.amount * thrtgrp^.readyrate
                          * (1.0 - thrtgrp^.attrition);
                 thrtgrp^.amount := (1.0 - thrtgrp^.attrition) * thrtgrp^.amount;
                 if thrtgrp^.amount < thrtgrp^.minimum then
                        thrtgrp^.amount := thrtgrp^.minimum;
                 end
                 else
                 avail := 0;

      {sof   days 1, 1+sofrq, 1+2*sofrq...}
              3: if (day mod sofrq = 1) and (thrtgrp^.readyrate > 0) then
                    begin
                    avail :=int(thrtgrp^.amount
                            * (1-thrtgrp^.readyrate)+0.5); {pre tgt attrit}
                    thrtgrp^.amount :=int(avail *
                           (1 - thrtgrp^.attrition)+0.5); {post tgt attrition}
                    end
                    else
                    avail := 0.0;

      {ssm}
              4: if (day mod ssmfrq = 1) and (thrtgrp^.readyrate > 0) then
                    begin
                    if thrtgrp^.amount > 0 then
                       begin
                       avail := int(thrtgrp^.amount*thrtgrp^.attrition+0.5);
                       if avail > 0 then
                          thrtgrp^.amount := thrtgrp^.amount - avail
                          else
                          if thrtgrp^.amount > 1 then
                             begin
                             avail := 1;
                             thrtgrp^.amount := thrtgrp^.amount - avail;
                             end
```

```
                        else
                        avail := 0.0;
                  end
                  else
                  avail := 0.0;
                end
               else
               avail := 0.0;

         else avail := 0.0;
      end;

      sorties[day,xt] := sorties[day,xt] + avail;
      if avail > 0 then
         itgt := front.first
         else
         itgt := nil;
      while itgt <> nil do
         begin
         if thrtgrp^.reaches(itgt) then
            itgt^.attack(avail,log,xt,day,nperiod,sprsdays,
                      thrtgrp^.name);
         if avail <= 0 then
            itgt := nil
            else
            itgt := itgt^.next;
         end;
      writeln(log,'...[',day:2,'] ...',thrtgrp^.name,
                               ' - unused ',avail:5:1);
      thrtgrp := thrtgrp^.next;
      end {xt loop};

{..................report process.................................}

   if ((day mod period) = 0) or (day = maxday) then
      begin
      itgt := front.first;
      while itgt <> nil do
         begin
         if (pos(itgt^.nation,countries) > 0) or (countries[1] = '*') then
            begin
            itgt^.pdreport(nperiod,day,maxfac,reportbl);
            end;
         itgt := itgt^.next;
         end;
      writeln;writeln('#Report for period ending day',day:3,' for countries
',countries);
      facil := ftable^.first;
      i := 0;
      writeln;writeln('CatCode       Facility      ',
         '    Onhand      Damage      Hits    Craters ');
      writeln('-------    -------------------',
         ' ----------  ----------  --------  --------');
      while facil <> nil do
         begin
         i := i + 1;
         write('[',facil^.code,'] ');
         write(facil^.name:22);
         write(rollup[i]:12:0);
         write(reportbl[i,1]:12:1);
         writeln(reportbl[i,2]:10:2,reportbl[i,3]:10:2 );
         for j := 1 to 3 do
            begin
            totals[i,j] := totals[i,j] + reportbl[i,j];
            reportbl[i,j] := 0.0;
            end;
         facil := facil^.next;
```

```
            end;
         end;

   {.......Installation damage reconstitution process..................}

         begin
         if (day mod reconst) = 0 then
            begin
            if reconstno > 0 then
               begin
               reconstno := reconstno - 1;
               writeln('===installations reconstituted at day ',day:5);
               itgt := front.first;
               while itgt <> nil do
                  begin
                  itgt^.update(nperiod,day,((day mod reconst)= 0));
                  itgt := itgt^.next;
                  end;
               end
               else
               reconst := 999;
            end;
   {...............................................................}

         if  ((day mod period) = 0) then nperiod := nperiod + 1;
         { writeln('---facility damage accumulated at day',day:5); }
         end;

      end {day loop};

   {..............simulation portion completed.......................}

      close(log);

      writeln;writeln('#Summary Report for entire period (',day:4,
                            'days) for countries ',countries);
      facil := ftable^.first;       i := 0;
      writeln;writeln('CatCode           Facility        ',
           '    Onhand      Damage      Hits     Craters ');
      writeln('-------   ------------------',
           ' ----------  ----------  --------  --------');
      while facil <> nil do
         begin
         i := i + 1;
         write('[',facil^.code,'] ');
         write(facil^.name:22);
         write(rollup[i]:12:0);
         write(totals[i,1]:12:1);
         writeln(totals[i,2]:10:2,totals[i,3]:10:2 );
         facil := facil^.next;
         end;
   minutes := elapsed(minutes);

   {..........................optional reports '!'.................}

   if repinstal then
      begin
      front.postmortem(maxfac,nperiod-1,period,ftable,countries);
      minutes := elapsed(minutes);
      dtable^.breakout(nperiod-1);
      minutes := elapsed(minutes);
      end;

   writeln;writeln('Sorties summary:...........');writeln;

   for j := 1 to nthrt do
      begin
```

```
          write('(T=',j:1,')');
          for i := 1 to maxday do
             begin
             write(sorties[i,j]:3:0);
             if (i mod period) = 0 then write(' |');
             end;
          writeln;
          end;

       writeln;write(' heap memory =',memavail:10);minutes := elapsed(0.0);
       end.
```

LAST PAGE OF APPENDIX C-3

Engineer Studies Center	A THREAT-BASED THEATER WAR DAMAGE METHODOLOGY	STUDY GIST CEESC-R-91-20

PRINCIPAL FINDINGS: Among the most difficult tasks confronting engineer and logistics planners is estimating war damage to infrastructure and installation facilities. The U.S. Army Engineer Studies Center (ESC) has been wrestling with this problem since the late 1970s while analyzing engineer requirements under various operational plans. The early assessments used different approaches to estimating war damage. The lack of uniformity and inability to reuse these disparate approaches prompted ESC to develop a general damage methodology that has the following advantages:

- *Threat-Based*: Damage is purposely constrained to the capability of threat forces. Damage from threat fighter, bomber, surface-to-surface missiles, and special operations forces can be assessed. Various operational attributes are also identified (e.g., range, ordnance load, readiness).

- *Scenario-Dependent*: The methodology requires that all installation targets and applicable enemy bases, or origins, be identified. While not a combat model, it can emulate changes in the theater disposition by varying attrition and readiness rates, as well as threat redeployments. These data would correspond to guidance from intelligence sources, operational plans, or wargamed results.

- *Detailed Results*: Results are calculated at installation/facility level. While rollup reports on user-defined time periods and installation groupings are available, the user can modify the software and access or portray even more detailed data.

- *Accessible and Adaptable*: ESC's implementation uses software that can run on any PC-compatible microcomputer. This makes the methodology as universally available as possible. Furthermore, the threat-dependent theater portion of the methodology employs an object-oriented model that greatly facilitates extensibility if additional features are desired.

SCOPE OF THE STUDY: The study consists of a main paper and three annexes. The main portion describes the rationale, assumptions, and operational features. (The methodology is essentially two-phased: the first requires the use of a detailed damage assessment computer program; the second utilizes a theater damage model developed by ESC.) The annexes document the input, output, and internal code of the theater model.

REPORT OBJECTIVE: The purpose of this document is to describe the background and features of ESC's threat-based war damage methodology, define data requirements, and present examples of input and output. A secondary objective is to promote the transfer and distribution of the methodology to defense organizations concerned with the problem.

BASIC APPROACH: ESC's objective was to develop a reasonable and reproducible, threat-based system to estimate facility damage across a theater. A two-phased approach evolved. A detailed installation-level damage model is used to generate a library of attack results (damage profiles). The

library is then one input to the deterministic theater damage assessment model, along with scenario specific information regarding threat capability and targets. The assessment model is more an allocation than an explicit damage model since its purpose is to distribute threat assets among theater targets according to defined target priorities, mission requirements, and sortie constraints. Damage is calculated by referencing the appropriate entry in the profile library. Therefore, it is not necessary to repeat the calculations already made in the detailed damage model. Theater damage thus becomes a process of allocating missions against identified targets or classes of targets and assessing the expected damage associated with that allotment.

REASONS FOR PERFORMING THE STUDY: In addition to their mission of constructing and maintaining the theater sustainment base, engineers are responsible for repairing or replacing facilities that are damaged. Planning for the expected amount and kinds of repair, however, is confounded by the vagaries of war. Theater wargames typically ignore most rear area installations, or estimate rear area damage in such broad (or parochial) terms as to be useless for repair estimates. Separate installation-level programs exist that model the effect of individually-targeted munitions and the resulting direct and collateral facility damage. But such programs usually examine one attack against one installation, and are too cumbersome and detailed to be useful at theater-level. ESC has developed a methodology that builds on the capabilities of these detailed installation-level models. An approach was formulated that utilizes the output of these high resolution models, the best available intelligence, and the estimates of theater-level enemy capability to project damage by facility, installation, and time. Having succeeded in implementing this approach, ESC sought to document the methodology and publicize its availability.

STUDY SPONSOR: Deputy Director for Plans and Resources, J-4, Joint Chiefs of Staff.

PERFORMING ORGANIZATION AND PRINCIPAL AUTHORS: The study was prepared by the US Army Engineer Studies Center. The principal author was Robert Halayko.

DTIC ACCESSION NUMBER OF FINAL STUDY: Pending.

COMMENTS AND SUGGESTIONS MAY BE SENT TO: Commander, U.S. Army Engineer Studies Center, Casey Building #2594, Fort Belvoir, Virginia 22060-5583.

START AND COMPLETION DATES OF STUDY: Starting Date: January 1991
Completion Date: June 1991